Anytime pmnk-landscapes

Generated by Doxygen 1.9.2

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 pmnk::EPS Struct Reference

Additive epsilon indicator.

`#include <functor.hpp>`

Public Member Functions

- constexpr **EPS** ()=default

  Construct a new EPS object.
- template<typename S = GASolution>
  constexpr double operator() (S const &s1, S const &s2) const noexcept

  Function call operator overload. Implements the indicator functionality.

4.1.1 Detailed Description

Additive epsilon indicator.

4.1.2 Member Function Documentation

4.1.2.1 operator()()

```
template<typename S = GASolution>
constexpr double pmnk::EPS::operator() (
            S const & s1,
            S const & s2 ) const  [inline], [constexpr], [noexcept]
```

Function call operator overload. Implements the indicator functionality.

Template Parameters

S	The type used to store an genetic algorithm (IBEA) solution

Parameters

s1	A solution to be evaluated.
s2	A solution to be evaluated.

Returns

double The value for the indicator.

The documentation for this struct was generated from the following file:

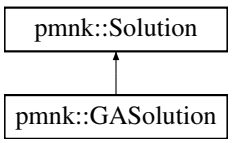
- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp

4.2 pmnk::GASolution Class Reference

Genetic algorithm solution wrapper (adds a fitness attribute to the Solution class)

`#include <solution.hpp>`

Inheritance diagram for pmnk::GASolution:



Public Member Functions

- **GASolution** ()=default

  Construct a new GASolution object.
- GASolution (Solution &&sol, double fitness)

  Construct a new GASolution object (move)
- GASolution (Solution &&sol)

  Construct a new GASolution object.
- constexpr double const & fitness () const

  Getter method for the GASolution's fitness value.
- void set_objv (ObjectiveVector &&objv)

  Set the objv object of the current solution.
- constexpr void set_fitness (double const fitness)

  Setter method for the fitness value.

Additional Inherited Members

4.2.1 Detailed Description

Genetic algorithm solution wrapper (adds a fitness attribute to the Solution class)

4.2.2 Constructor & Destructor Documentation

4.2.2.1 GASolution() [1/2]

```
pmnk::GASolution::GASolution (
            Solution && sol,
            double fitness )  [inline]
```

Construct a new GASolution object (move)

Parameters

sol	A rvalue reference to a solution.
fitness	The fitness value of the solution.

4.2.2.2 GASolution() [2/2]

```
pmnk::GASolution::GASolution (
            Solution && sol )  [inline]
```

Construct a new GASolution object.

Parameters

sol	A rvalue reference to a solution. (fitness value defaults to 0)

4.2.3 Member Function Documentation

4.2.3.1 fitness()

```
constexpr double const & pmnk::GASolution::fitness ( ) const  [inline], [constexpr]
```

Getter method for the GASolution's fitness value.

Returns

constexpr double const& The fitness value.

4.2.3.2 set_fitness()

```
constexpr void pmnk::GASolution::set_fitness (
            double const fitness )  [inline], [constexpr]
```

Setter method for the fitness value.

Parameters

fitness	The fitness value.

4.2.3.3 set_objv()

```
void pmnk::GASolution::set_objv (
            ObjectiveVector && objv )  [inline]
```

Set the objv object of the current solution.

Parameters

objv	A rvalue reference to the objective vector.

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/solution.hpp

4.3 pmnk::GSEMO Class Reference

Wrapper class for GSEMO.

```
#include <gsemo.hpp>
```

Public Member Functions

- template<typename Str = std::string, typename Ref = ObjectiveVector>
  GSEMO (Str &&instance, unsigned int const seed, std::ostream &os, Ref &&ref)

  Construct a new GSEMO object.
- template<typename Str = std::string>
  GSEMO (Str &&instance, unsigned int const seed, std::ostream &os)

Construct a new GSEMO object.

- template<typename Str = std::string, typename Ref = ObjectiveVector>
  GSEMO (Str &&instance, unsigned int const seed, Ref &&ref)

  Construct a new GSEMO object.
- template<typename Str = std::string>
  GSEMO (Str &&instance, unsigned int const seed)

  Construct a new GSEMO object.
- template<typename Str = std::string, typename Ref = ObjectiveVector>
  GSEMO (Str &&instance, Ref &&ref)

  Construct a new GSEMO object.
- template<typename Str = std::string>
  GSEMO (Str &&instance)

  Construct a new GSEMO object.
- std::vector< Solution > const & solutions () const

  Getter for the vector of solutions found by this algorithm.
- void run (std::size_t maxeval)

  GSEMO implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.

Public Attributes

- RMNKEval **eval**

4.3.1 Detailed Description

Wrapper class for GSEMO.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 GSEMO() [1/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::GSEMO::GSEMO (
            Str && instance,
            unsigned int const seed,
            std::ostream & os,
            Ref && ref )  [inline]
```

Construct a new GSEMO object.

Template Parameters

Str	the type used to store the instance path
Ref	the type used to store the reference point of the hvobj obj

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in GSEMO
os	The name of the output file where the standard output stream should be redirected
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

4.3.2.2 GSEMO() [2/6]

```
template<typename Str = std::string>
pmnk::GSEMO::GSEMO (
            Str && instance,
            unsigned int const seed,
            std::ostream & os )  [inline]
```

Construct a new GSEMO object.

Template Parameters

Str	the type used to store the instance path

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in GSEMO
os	The name of the output file where the standard output stream should be redirected

4.3.2.3 GSEMO() [3/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::GSEMO::GSEMO (
            Str && instance,
            unsigned int const seed,
            Ref && ref )  [inline]
```

Construct a new GSEMO object.

Template Parameters

Str	the type used to store the instance path
Ref	the type used to store the reference point of the hvobj obj

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in GSEMO
os	The name of the output file where the standard output stream should be redirected
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

4.3.2.4 GSEMO() [4/6]

```
template<typename Str = std::string>
pmnk::GSEMO::GSEMO (
            Str && instance,
            unsigned int const seed )  [inline]
```

Construct a new GSEMO object.

Template Parameters

Str	the type used to store the instance path

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in GSEMO

4.3.2.5 GSEMO() [5/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::GSEMO::GSEMO (
            Str && instance,
            Ref && ref )  [inline], [explicit]
```

Construct a new GSEMO object.

Template Parameters

Str	the type used to store the instance path
Ref	the type used to store the reference point of the hvobj obj

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

4.3.2.6 GSEMO() [6/6]

```
template<typename Str = std::string>
pmnk::GSEMO::GSEMO (
            Str && instance )  [inline], [explicit]
```

Construct a new GSEMO object.

Template Parameters

Str	the type used to store the instance path

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script

4.3.3 Member Function Documentation

4.3.3.1 run()

```
void pmnk::GSEMO::run (
            std::size_t maxeval )  [inline]
```

GSEMO implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.

Parameters

maxeval	The maximum number of evaluations performed by GSEMO (stopping criterion)

4.3.3.2 solutions()

```
std::vector< Solution > const & pmnk::GSEMO::solutions ( ) const  [inline]
```

Getter for the vector of solutions found by this algorithm.

Returns

std::vector<Solution> const& Read-Only reference to a vector of solutions found by the GSEMO algorithm.

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/GSEMO/gsemo.hpp

4.4 pmnk::hvobj< T > Class Template Reference

Implementation of an API that supports among others, set/point hypervolume calculations (using the WFG algorithm)

`#include <wfg.hpp>`

Public Types

- using **hv_type** = T
- using **ovec_type** = std::vector< hv_type >
- using **set_type** = std::vector< ovec_type >

Public Member Functions

- constexpr **hvobj** (ovec_type const &r)
- constexpr **hvobj** (hvobj const &other)=default
- constexpr **hvobj** (hvobj &&other) noexcept=default
- constexpr auto **value** () const

  Get the current hypervolume value.
- template<typename V >
  constexpr auto **contribution** (V const &v) const

  Get the contribution of a new vector w.r.t. to the current set.
- template<typename V >
  constexpr auto **insert** (V &&v)

  Inserts a new objective vector and returns its contribution.
- template<typename V >
  constexpr auto remove (V const &v)

4.4.1 Detailed Description

template<typename T>
class pmnk::hvobj< T >

Implementation of an API that supports among others, set/point hypervolume calculations (using the WFG algorithm)

4.4.2 Member Function Documentation

4.4.2.1 remove()

```
template<typename T >
template<typename V >
constexpr auto pmnk::hvobj< T >::remove (
            V const & v )  [inline], [constexpr]
```

Removes a objective vector and returns its contribution (i.e. the lost hv) or -1.0 if no objective vector was found.

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/wfg.hpp

4.5 pmnk::IBEA Class Reference

Wrapper class for IBEA.

```
#include <ibea.hpp>
```

Public Member Functions

- template<typename Str = std::string, typename Ref = ObjectiveVector>
  IBEA (Str &&instance, unsigned int const seed, std::ostream &os, Ref &&ref)

  Construct a new IBEA object.
- template<typename Str = std::string>
  IBEA (Str &&instance, unsigned int const seed, std::ostream &os)

  Construct a new IBEA object.
- template<typename Str = std::string, typename Ref = ObjectiveVector>
  IBEA (Str &&instance, unsigned int const seed, Ref &&ref)

  Construct a new IBEA object.
- template<typename Str = std::string>
  IBEA (Str &&instance, unsigned int const seed)

  Construct a new IBEA object.
- template<typename Str = std::string, typename Ref = ObjectiveVector>
  IBEA (Str &&instance, Ref &&ref)

  Construct a new IBEA object.
- template<typename Str = std::string>
  IBEA (Str &&instance)

  Construct a new IBEA object.
- std::vector< GASolution > const & solutions () const

  Getter for the vector of solutions found by this algorithm.
- template<typename I , typename S , typename M , typename C >
  void run (std::size_t const maxeval, std::size_t const population_max_size, std::size_t const max_↩ generations, double const scaling_factor, I &&indicator, C &&crossover_method, M &&mutation_method, S &&selection_method, bool adaptive)

  IBEA implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.

Public Attributes

- RMNKEval **eval**

4.5.1 Detailed Description

Wrapper class for IBEA.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 IBEA() [1/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::IBEA::IBEA (
            Str && instance,
            unsigned int const seed,
            std::ostream & os,
            Ref && ref )  [inline]
```

Construct a new IBEA object.

Template Parameters

Str	the type used to store the instance path
Ref	the type used to store the reference point of the hvobj obj

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in IBEA
os	The name of the output file where the standard output stream should be redirected
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

4.5.2.2 IBEA() [2/6]

```
template<typename Str = std::string>
pmnk::IBEA::IBEA (
            Str && instance,
            unsigned int const seed,
            std::ostream & os )  [inline]
```

Construct a new IBEA object.

Template Parameters

Str	the type used to store the instance path

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in IBEA
os	The name of the output file where the standard output stream should be redirected

4.5.2.3 IBEA() [3/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::IBEA::IBEA (
            Str && instance,
            unsigned int const seed,
            Ref && ref )  [inline]
```

Construct a new IBEA object.

Template Parameters

Str	the type used to store the instance path
Ref	the type used to store the reference point of the hvobj obj

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in IBEA
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

4.5.2.4 IBEA() [4/6]

```
template<typename Str = std::string>
pmnk::IBEA::IBEA (
            Str && instance,
            unsigned int const seed )  [inline]
```

Construct a new IBEA object.

Template Parameters

Str	the type used to store the instance path

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in IBEA

4.5.2.5 IBEA() [5/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::IBEA::IBEA (
            Str && instance,
            Ref && ref )  [inline], [explicit]
```

Construct a new IBEA object.

Template Parameters

Str	the type used to store the instance path
Ref	the type used to store the reference point of the hvobj obj

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in IBEA
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

4.5.2.6 IBEA() [6/6]

```
template<typename Str = std::string>
pmnk::IBEA::IBEA (
            Str && instance )  [inline], [explicit]
```

Construct a new IBEA object.

Template Parameters

Str	the type used to store the instance path

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script

4.5.3 Member Function Documentation

4.5.3.1 run()

```
template<typename I , typename S , typename M , typename C >
void pmnk::IBEA::run (
            std::size_t const maxeval,
            std::size_t const population_max_size,
            std::size_t const max_generations,
            double const scaling_factor,
            I && indicator,
            C && crossover_method,
            M && mutation_method,
            S && selection_method,
            bool adaptive )  [inline]
```

IBEA implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.

Template Parameters

I	The type used to store an IBEA indicator
S	The type used to store an IBEA selection operator
M	The type used to store and IBEA mutation operator
C	The type used to store and IBEA crossover operator

Parameters

maxeval	The maximum number of evaluations performed by the algorithms (stopping criterion)
population_max_size	The maximum population size
max_generations	The maximum number of generations
scaling_factor	The scaling factor
indicator	The indicator to be used by the IBEA indicator operator
crossover_method	The crossover method considered by the IBEA mutation operator
mutation_method	The mutation method considered by the IBEA mutation operator
selection_method	The selection method considered by the IBEA selection operator
adaptive	boolean indicative of version of IBEA to be used. If true use adaptive version of (A-IBEA) else use (B-IBEA)

4.5.3.2 solutions()

```
std::vector< GASolution > const & pmnk::IBEA::solutions ( ) const  [inline]
```

Getter for the vector of solutions found by this algorithm.

Returns

std::vector<Solution> const& Read-Only reference to a vector of solutions found by the IBEA.

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/ibea.hpp

4.6 pmnk::IHD< R > Struct Template Reference

Hypervolume Based IBEA indicator.

```
#include <functor.hpp>
```

Public Member Functions

- constexpr IHD (R &&ref)
  Construct a new IHD object.
- template<typename S = GASolution>
  double operator() (S const &s1, S const &s2) const
  Function call operator overload. Implements the indicator functionality.

Public Attributes

- R **m_ref**

4.6.1 Detailed Description

template<typename R = ObjectiveVector>
struct pmnk::IHD< R >

Hypervolume Based IBEA indicator.

Template Parameters

R	The IBEA reference point type.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 IHD()

```
template<typename R = ObjectiveVector>
constexpr pmnk::IHD< R >::IHD (
            R && ref )  [inline], [explicit], [constexpr]
```

Construct a new IHD object.

Parameters

ref	The reference point used for indicator calculation.

4.6.3 Member Function Documentation

4.6.3.1 operator()()

```
template<typename R = ObjectiveVector>
template<typename S = GASolution>
double pmnk::IHD< R >::operator() (
            S const & s1,
            S const & s2 ) const  [inline]
```

Function call operator overload. Implements the indicator functionality.

Template Parameters

S	The type used to store an genetic algorithm (IBEA) solution

Parameters

s1	A solution to be evaluated.
s2	A solution to be evaluated.

Returns

double The value for the indicator.

The documentation for this struct was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp

4.7 pmnk::KWayTournamentSelection< RNG > Struct Template Reference

KWayTournamentSelection operator.

`#include <functor.hpp>`

Public Member Functions

- constexpr KWayTournamentSelection (std::size_t const tournament_size, std::size_t const matting_pool_↩ size, RNG &rng)

  Construct a new KWayTournamentSelection object.
- template<typename S = GASolution>
  std::vector< S > operator() (std::vector< S > const &population) noexcept

  Function call operator overload. Implements the selection operator functionality.

Public Attributes

- std::size_t **m_tournament_size**
- std::size_t **m_matting_pool_size**
- RNG **m_rng**

4.7.1 Detailed Description

template<typename RNG>
struct pmnk::KWayTournamentSelection< RNG >

KWayTournamentSelection operator.

Template Parameters

RNG	The type for the random number generator object.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 KWayTournamentSelection()

```
template<typename RNG >
constexpr pmnk::KWayTournamentSelection< RNG >::KWayTournamentSelection (
            std::size_t const tournament_size,
            std::size_t const matting_pool_size,
            RNG & rng )  [inline], [constexpr]
```

Construct a new KWayTournamentSelection object.

Parameters

tournament_size	The size of the tournament (K)
matting_pool_size	The the maximum number of individuals allowed in the matting pool.
rng	The random number generator objects.

4.7.3 Member Function Documentation

4.7.3.1 operator()()

```
template<typename RNG >
template<typename S = GASolution>
std::vector< S > pmnk::KWayTournamentSelection< RNG >::operator() (
            std::vector< S > const & population )  [inline], [noexcept]
```

Function call operator overload. Implements the selection operator functionality.

Template Parameters

S	The type used to store an genetic algorithm (IBEA) solution

Parameters

population	The population from which the individuals will be selected.

Returns

std::vector<S> The matting pool obtain as a result from the selection of the individuals of the population.

The documentation for this struct was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp

4.8 pmnk::NPointCrossover< RNG > Struct Template Reference

N-Point crossover operator.

`#include <functor.hpp>`

Public Member Functions

- constexpr NPointCrossover (std::size_t const crossover_points, double const crossover_probability, RNG &rng)

  Construct a new NPointCrossover.
- template<typename S = GASolution>
  void operator() (S &s1, S &s2) noexcept

  Function call operator overload. Implements the crossover operator functionality.

Public Attributes

- std::size_t **m_crossover_points**
- double **m_crossover_probability**
- RNG **m_rng**
- std::uniform_real_distribution< double > **m_distrib**

4.8.1 Detailed Description

template<typename RNG>
struct pmnk::NPointCrossover< RNG >

N-Point crossover operator.

Template Parameters

RNG	The type for the random number generator object.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 NPointCrossover()

```
template<typename RNG >
constexpr pmnk::NPointCrossover< RNG >::NPointCrossover (
            std::size_t const crossover_points,
            double const crossover_probability,
            RNG & rng )  [inline], [constexpr]
```

Construct a new NPointCrossover.

Parameters

crossover_points	Number of crossover points considered by this operator.
crossover_probability	Crossover probability considered by this operator.
rng	The random number generator object instance.

4.8.3 Member Function Documentation

4.8.3.1 operator()()

```
template<typename RNG >
template<typename S = GASolution>
```

```
void pmnk::NPointCrossover< RNG >::operator() (
            S & s1,
            S & s2 )  [inline], [noexcept]
```

Function call operator overload. Implements the crossover operator functionality.

Parameters

s1	A solution to be recombinated.
s2	A solution to be recombinated.

The documentation for this struct was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp

4.9 pmnk::PLS Class Reference

Wrapper class for PLS.

```
#include <pls.hpp>
```

Public Member Functions

- template<typename Str = std::string, typename Ref = ObjectiveVector>
  PLS (Str &&instance, unsigned int seed, std::ostream &os, Ref &&ref)

  Construct a new PLS object.
- template<typename Str = std::string>
  PLS (Str &&instance, unsigned int seed, std::ostream &os)

  Construct a new PLS object.
- template<typename Str = std::string, typename Ref = ObjectiveVector>
  PLS (Str &&instance, unsigned int seed, Ref &&ref)

  Construct a new PLS object.
- template<typename Str = std::string>
  PLS (Str &&instance, unsigned int seed)

  Construct a new PLS object.
- template<typename Str = std::string, typename Ref = ObjectiveVector>
  PLS (Str &&instance, Ref &&ref)

  Construct a new PLS object.
- template<typename Str = std::string>
  PLS (Str &&instance)

  Construct a new PLS object.
- std::vector< Solution > const solutions () const

  Getter for the vector of solutions found by this algorithm.
- std::vector< Solution > const non_visited_solutions () const

  Getter for the vector of solutions found by this algorithm that were not visited in the process of local search.
- void run (std::size_t maxeval, PLSAcceptanceCriterion const acceptance_criterion, PLSExplorationCriterion const neighborhood_exploration)

  PLS implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.

Public Attributes

- RMNKEval **eval**

4.9.1 Detailed Description

Wrapper class for PLS.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 PLS() [1/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::PLS::PLS (
            Str && instance,
            unsigned int seed,
            std::ostream & os,
            Ref && ref )  [inline]
```

Construct a new PLS object.

Template Parameters

Str	the type used to store the instance path
Ref	the type used to store the reference point of the hvobj obj

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in PLS
os	The name of the output file where the standard output stream should be redirected
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

4.9.2.2 PLS() [2/6]

```
template<typename Str = std::string>
pmnk::PLS::PLS (
            Str && instance,
            unsigned int seed,
            std::ostream & os )  [inline]
```

Construct a new PLS object.

Template Parameters

Str	the type used to store the instance path

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in PLS
os	The name of the output file where the standard output stream should be redirected

4.9.2.3 PLS() [3/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::PLS::PLS (
            Str && instance,
            unsigned int seed,
            Ref && ref )  [inline]
```

Construct a new PLS object.

Template Parameters

Str	the type used to store the instance path
Ref	the type used to store the reference point of the hvobj obj

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in PLS
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

4.9.2.4 PLS() [4/6]

```
template<typename Str = std::string>
pmnk::PLS::PLS (
            Str && instance,
            unsigned int seed )  [inline]
```

Construct a new PLS object.

Template Parameters

Str	the type used to store the instance path

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
seed	The seed used by the pseudo random number generator used in PLS

4.9.2.5 PLS() [5/6]

```
template<typename Str = std::string, typename Ref = ObjectiveVector>
pmnk::PLS::PLS (
            Str && instance,
            Ref && ref )  [inline]
```

Construct a new PLS object.

Template Parameters

Str	the type used to store the instance path
Ref	the type used to store the reference point of the hvobj obj

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

4.9.2.6 PLS() [6/6]

```
template<typename Str = std::string>
pmnk::PLS::PLS (
            Str && instance )  [inline]
```

Construct a new PLS object.

Template Parameters

Str	the type used to store the instance path

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script

4.9.3 Member Function Documentation

4.9.3.1 non_visited_solutions()

```
std::vector< Solution > const pmnk::PLS::non_visited_solutions ( ) const  [inline]
```

Getter for the vector of solutions found by this algorithm that were not visited in the process of local search.

Returns

std::vector<Solution> const& Read-Only reference to a vector of non visited solutions produced by PLS.

4.9.3.2 run()

```
void pmnk::PLS::run (
            std::size_t maxeval,
            PLSAcceptanceCriterion const acceptance_criterion,
            PLSExplorationCriterion const neighborhood_exploration )  [inline]
```

PLS implementation runner. This effectively starts the algorithm and runs it until the maximum number of evaluations has been reached.

Parameters

maxeval	The maximum number of evaluations performed by PLS (stopping criterion)
acceptance_criterion	The PLS algorithm solution acceptance criterion
neighborhood_exploration	The PLS algorithm solution exploration criterion

4.9.3.3 solutions()

```
std::vector< Solution > const pmnk::PLS::solutions ( ) const  [inline]
```

Getter for the vector of solutions found by this algorithm.

Returns

std::vector<Solution> const& Read-Only reference to a vector of visited solutions produced by PLS.

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/PLS/pls.hpp

4.10 pmnk::RMNKEval Class Reference

rmnk_landscapes instance evaluator

`#include <rMNKEval.hpp>`

Public Member Functions

- **RMNKEval** (const char *_fileName)
- void **eval** (std::vector< bool > &_solution, std::vector< double > &_objVec)
- unsigned **getM** ()
- unsigned **getN** ()
- unsigned **getK** ()
- double **getRho** ()

Protected Member Functions

- virtual void **load** (const char *_fileName)
- void **init** ()
- void **loadLinks** (std::fstream &_file)
- void **loadTables** (std::fstream &_file)
- double **evalNK** (unsigned _numObj, std::vector< bool > &_sol)
- unsigned int **sigma** (unsigned _numObj, std::vector< bool > &_sol, int _i)

Protected Attributes

- double **rho**
- unsigned **M**
- unsigned **N**
- unsigned **K**
- double *** **tables**
- unsigned *** **links**

4.10.1 Detailed Description

rmnk_landscapes instance evaluator

The documentation for this class was generated from the following file:

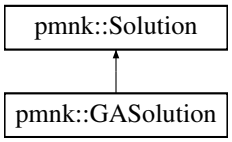
- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/rMNKEval.hpp

4.11 pmnk::Solution Class Reference

Standard solution class.

`#include <solution.hpp>`

Inheritance diagram for pmnk::Solution:



Public Member Functions

- Solution (Solution const &other)=default

  Construct a new Solution object (copy constructor)
- Solution (Solution &&other)=default

  Construct a new Solution object (move constructor)
- Solution & operator= (Solution const &other)=default

  Defaulted copy assignment operator.
- Solution & operator= (Solution &&other)=default

  Defaulted move assignment operator.
- Solution (RMNKEval &rmnk, DecisionVector const &decision)

  Construct a new Solution object.
- Solution (RMNKEval &rmnk, DecisionVector &&decision)

  Construct a new Solution object.
- DecisionVector const & decision_vector () const

  Getter for the solution's decision vector.
- ObjectiveVector const & objective_vector () const

  Getter for the solution's objective vector.
- std::size_t size () const noexcept

  Getter for the solution's decision/objective vector size.
- DominanceType dominance (Solution const &solution) const

  Calculate the objective dominance type of this solution with respect to another.
- DecisionVector::reference operator[ ] (std::size_t const i)

  Array Indexing operator overload. This operator provides a way to access the i-th bit in the solution's decision vector implementation.
- void eval (RMNKEval &rmnk)

  Wrapper method that calls the RMNK solution evaluator suplied on the solution's decision vector, calculating the respective objective vector.

Static Public Member Functions

- template<typename RNG >
  static Solution random_solution (RMNKEval &eval, RNG &generator)

  Build and evaluate a new random solution object.
- template<typename RNG >
  static Solution uniform_bit_flip_solution (RMNKEval &eval, RNG &generator, Solution const &original)

  Build and evaluate a new Random solution object that results from a mutation in another solution's decision vector bit representation.
- static std::vector< Solution > neighborhood_solutions (RMNKEval &eval, Solution const &original)

  Calculate all the neighboor solutions of the current one.

Protected Attributes

- DecisionVector **m_decision**
- ObjectiveVector **m_objective**

Friends

- std::ostream & operator<< (std::ostream &os, Solution const &solution)

  Extraction operator overload for this object.

4.11.1 Detailed Description

Standard solution class.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Solution() [1/4]

```
pmnk::Solution::Solution (
            Solution const & other )  [default]
```

Construct a new Solution object (copy constructor)

Parameters

other	A const lvalue reference to a solution to be coppied

4.11.2.2 Solution() [2/4]

```
pmnk::Solution::Solution (
            Solution && other )  [default]
```

Construct a new Solution object (move constructor)

Parameters

other	A rvalue reference to a solution to be moved

4.11.2.3 Solution() [3/4]

```
pmnk::Solution::Solution (
            RMNKEval & rmnk,
            DecisionVector const & decision )  [inline]
```

Construct a new Solution object.

Parameters

rmnk	A lvalue reference to the RMNK instance evaluator.
decision	The solution's decision vector.

4.11.2.4 Solution() [4/4]

```
pmnk::Solution::Solution (
            RMNKEval & rmnk,
            DecisionVector && decision )  [inline]
```

Construct a new Solution object.

Parameters

rmnk	A lvalue reference to the RMNK instance evaluator.
decision	The solution's decision vector.

4.11.3 Member Function Documentation

4.11.3.1 decision_vector()

```
DecisionVector const & pmnk::Solution::decision_vector ( ) const  [inline]
```

Getter for the solution's decision vector.

Returns

DecisionVector const& A Read-Only reference to the solution's decision vector.

4.11.3.2 dominance()

```
DominanceType pmnk::Solution::dominance (
            Solution const & solution ) const  [inline]
```

Calculate the objective dominance type of this solution with respect to another.

Parameters

solution	A solution whose dominance type of this will be tested against.

Returns

DominanceType The solution's dominance type

4.11.3.3 eval()

```
void pmnk::Solution::eval (
            RMNKEval & rmnk )  [inline]
```

Wrapper method that calls the RMNK solution evaluator suplied on the solution's decision vector, calculating the respective objective vector.

Parameters

rmnk	The RMNK instance evaluator instance that provides the method used for solution evaluation.

4.11.3.4 neighborhood_solutions()

```
static std::vector< Solution > pmnk::Solution::neighborhood_solutions (
            RMNKEval & eval,
            Solution const & original )  [inline], [static]
```

Calculate all the neighboor solutions of the current one.

Parameters

eval	The instance evaluator object.
generator	The random number generator object

Returns

std::vector<Solution> A vector of solutions containing the current solution neighboor solutions.

4.11.3.5 objective_vector()

```
ObjectiveVector const & pmnk::Solution::objective_vector ( ) const  [inline]
```

Getter for the solution's objective vector.

Returns

ObjectiveVector const& A Read-Only reference to the solution's objective vector.

4.11.3.6 operator=() [1/2]

```
Solution & pmnk::Solution::operator= (
            Solution && other )  [default]
```

Defaulted move assignment operator.

Parameters

other	A rvalue reference to solution to be coppied

Returns

Solution& A refernce to the solution to be assigned.

4.11.3.7 operator=() [2/2]

```
Solution & pmnk::Solution::operator= (
            Solution const & other )  [default]
```

Defaulted copy assignment operator.

Parameters

other	A const lvalue reference to a solution to be coppied.

Returns

Solution& A lvalue reference to the solution to be assigned.

4.11.3.8 operator[]()

```
DecisionVector::reference pmnk::Solution::operator[] (
            std::size_t const i )  [inline]
```

Array Indexing operator overload. This operator provides a way to access the i-th bit in the solution's decision vector implementation.

Parameters

i	The index of the element to be accessed.

Returns

DecisionVector::reference A lvalue reference to value contained in the accessed index.

4.11.3.9 random_solution()

```
template<typename RNG >
static Solution pmnk::Solution::random_solution (
            RMNKEval & eval,
            RNG & generator )  [inline], [static]
```

Build and evaluate a new random solution object.

Template Parameters

RNG	The type for the random number generator object.

Parameters

eval	The instance evaluator object.
generator	The random number generator object

Returns

Solution A new Solution object containing a random solution.

4.11.3.10 size()

```
std::size_t pmnk::Solution::size ( ) const  [inline], [noexcept]
```

Getter for the solution's decision/objective vector size.

Returns

std::size_t The size of this solution's decision/objective vector.

4.11.3.11 uniform_bit_flip_solution()

```
template<typename RNG >
static Solution pmnk::Solution::uniform_bit_flip_solution (
            RMNKEval & eval,
            RNG & generator,
            Solution const & original )  [inline], [static]
```

Build and evaluate a new Random solution object that results from a mutation in another solution's decision vector bit representation.

Template Parameters

RNG	The type for the random number generator object.

Parameters

eval	The instance evaluator object.
generator	The random number generator object
original	The parent solution to be mutated.

Returns

Solution A new Solution object containing a random solution.

4.11.4 Friends And Related Function Documentation

4.11.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            Solution const & solution )  [friend]
```

Extraction operator overload for this object.

Parameters

os	The output stream to where the solution string representation will be redirected.
solution	The solution whose representation will be inserted in the output stream

Returns

std::ostream& A output stream object with the solution's data appended to it.

The documentation for this class was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/solution.hpp

4.12 pmnk::UniformCrossover< RNG > Struct Template Reference

Uniform Crossover operator.

`#include <functor.hpp>`

Public Member Functions

- constexpr UniformCrossover (double crossover_probability, RNG &rng)

  Construct a new UniformCrossover object.
- template<typename S = GASolution>
  constexpr void operator() (S &s1, S &s2) noexcept

  Function call operator overload. Implements the crossover operator functionality.

Public Attributes

- double **m_crossover_probability**
- RNG **m_rng**
- std::bernoulli_distribution **m_distrib**

4.12.1 Detailed Description

template<typename RNG>
struct pmnk::UniformCrossover< RNG >

Uniform Crossover operator.

Template Parameters

RNG	The type for the random number generator object.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 UniformCrossover()

```
template<typename RNG >
constexpr pmnk::UniformCrossover< RNG >::UniformCrossover (
            double crossover_probability,
            RNG & rng )  [inline], [constexpr]
```

Construct a new UniformCrossover object.

Parameters

crossover_probability	Crossover probability considered in this operator.
rng	The random number generator object instance.

4.12.3 Member Function Documentation

4.12.3.1 operator()()

```
template<typename RNG >
template<typename S = GASolution>
constexpr void pmnk::UniformCrossover< RNG >::operator() (
            S & s1,
            S & s2 )  [inline], [constexpr], [noexcept]
```

Function call operator overload. Implements the crossover operator functionality.

Template Parameters

S	The type used to store an genetic algorithm (IBEA) solution

Parameters

s1	A solution to be recombinated.
s2	A solution to be recombinated.

The documentation for this struct was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp

4.13 pmnk::UniformMutation< RNG > Struct Template Reference

Uniform Mutation operator.

```
#include <functor.hpp>
```

Public Member Functions

- constexpr UniformMutation (double const mutation_probability, RNG &rng)

  Construct a new UniformMutation object.
- template<typename S = GASolution>
  constexpr void operator() (S &s) noexcept

  Function call operator overload. Implements the crossover operator functionality.

Public Attributes

- double **m_mutation_probability**
- RNG **m_rng**
- std::uniform_real_distribution< double > **m_distrib**

4.13.1 Detailed Description

template<typename RNG>
struct pmnk::UniformMutation< RNG >

Uniform Mutation operator.

Template Parameters

RNG	The type for the random number generator object.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 UniformMutation()

```
template<typename RNG >
constexpr pmnk::UniformMutation< RNG >::UniformMutation (
            double const mutation_probability,
            RNG & rng )  [inline], [constexpr]
```

Construct a new UniformMutation object.

Parameters

mutation_probability	Mutation probability considered in this operator.
rng	The random number generator object instance.

4.13.3 Member Function Documentation

4.13.3.1 operator()()

```
template<typename RNG >
template<typename S = GASolution>
constexpr void pmnk::UniformMutation< RNG >::operator() (
            S & s )  [inline], [constexpr], [noexcept]
```

Function call operator overload. Implements the crossover operator functionality.

Template Parameters

S	The type used to store an genetic algorithm (IBEA) solution

Parameters

s	A solution to be mutated.

The documentation for this struct was generated from the following file:

- /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp

Chapter 5

File Documentation

5.1 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/GSEMO/gsemo.hpp File Reference

GSEMO (Global Simple Multi-objective Optimizer) algorithm implementation.

```
#include <algorithm>
#include <csignal>
#include <iomanip>
#include <random>
#include "../Utils/solution.hpp"
#include "../Utils/utils.hpp"
#include "../Utils/wfg.hpp"
```

Classes

- class pmnk::GSEMO

  Wrapper class for GSEMO.

5.1.1 Detailed Description

GSEMO (Global Simple Multi-objective Optimizer) algorithm implementation.

Author

Pedro Rodrigues (pedror@student.dei.uc.pt)

Alexandre Jesus (ajesus@dei.uc.pt)

Version

0.1.0

Date

13-09-2021

Copyright

Copyright (c) 2021

5.2 gsemo.hpp

Go to the documentation of this file.

```
1
13 #ifndef GSEMO_HPP
14 #define GSEMO_HPP
15
16 #include <algorithm>
17 #include <csignal>
18 #include <iomanip>
19 #include <random>
20
21 #include "../Utils/solution.hpp"
22 #include "../Utils/utils.hpp"
23 #include "../Utils/wfg.hpp"
24
25 namespace pmnk {
26
28 class GSEMO {
29  public:
30   RMNKEval eval;
31
32  private:
33
34   std::mt19937 m_generator;
35   std::ostream &m_os;
36
37   hvobj<typename ObjectiveVector::value_type> m_hvo;
38   std::vector<Solution> m_solutions;
39
40  public:
41
53   template <typename Str = std::string, typename Ref = ObjectiveVector>
54   GSEMO(Str &&instance, unsigned int const seed, std::ostream &os, Ref &&ref)
55       : eval(std::forward<Str>(instance).c_str())
56       , m_generator(seed)
57       , m_os(os)
58       , m_hvo(std::forward<Ref>(ref)) {}
59
60
70   template <typename Str = std::string>
71   GSEMO(Str &&instance, unsigned int const seed, std::ostream &os)
72       : eval(std::forward<Str>(instance).c_str())
73       , m_generator(seed)
74       , m_os(os)
75       , m_hvo(ObjectiveVector(eval.getM(), 0)) {}
76
88   template <typename Str = std::string, typename Ref = ObjectiveVector>
89   GSEMO(Str &&instance, unsigned int const seed, Ref &&ref)
90       : GSEMO(std::forward<Str>(instance), seed, std::cout, std::forward<Ref>(ref)) {}
91
92
101   template <typename Str = std::string>
102   GSEMO(Str &&instance, unsigned int const seed)
103       : GSEMO(std::forward<Str>(instance), seed, std::cout) {}
104
114   template <typename Str = std::string, typename Ref = ObjectiveVector>
115   explicit GSEMO(Str &&instance, Ref &&ref)
116       : GSEMO(std::forward<Str>(instance), std::random_device()(), std::cout, std::forward<Ref>(ref)) {}
117
118
126   template <typename Str = std::string>
127   explicit GSEMO(Str &&instance)
128       : GSEMO(std::forward<Str>(instance), std::random_device()(), std::cout) {}
129
136   std::vector<Solution> const &solutions() const {
137     return m_solutions;
138   }
139
146   void run(std::size_t maxeval) {
147     auto rand_solution = Solution::random_solution(eval, m_generator);
148     m_hvo.insert(rand_solution.objective_vector());
149
150     add_non_dominated(m_solutions, std::move(rand_solution));
151     m_os << "evaluation,hypervolume\n";
152     m_os << std::setprecision(12) << 0 << "," << m_hvo.value() << "\n";
153
154     for (std::size_t i = 0; i < maxeval; ++i) {
155       std::uniform_int_distribution<std::size_t> randint(0, m_solutions.size() - 1);
156
157       std::size_t index = randint(m_generator);
158       auto solution = Solution::uniform_bit_flip_solution(eval, m_generator, m_solutions[index]);
159
160       auto sov = solution.objective_vector();
161       if (add_non_dominated(m_solutions, std::move(solution))) {
```

```
162         m_hvo.insert(sov);
163         m_os « std::setprecision(12) « i + 1 « "," « m_hvo.value() « "\n";
164       }
165     }
166   }
167 };
168 }  // namespace pmnk
169 #endif  //GSEMO_HPP
```

5.3 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/functor.hpp File Reference

Implementation of IBEA operators (using functors)

```
#include <iostream>
#include <random>
#include "../Utils/solution.hpp"
#include "../Utils/wfg.hpp"
```

Classes

- struct pmnk::IHD< R >
  Hypervolume Based IBEA indicator.
- struct pmnk::EPS
  Additive epsilon indicator.
- struct pmnk::NPointCrossover< RNG >
  N-Point crossover operator.
- struct pmnk::UniformCrossover< RNG >
  Uniform Crossover operator.
- struct pmnk::UniformMutation< RNG >
  Uniform Mutation operator.
- struct pmnk::KWayTournamentSelection< RNG >
  KWayTournamentSelection operator.

5.3.1 Detailed Description

Implementation of IBEA operators (using functors)

Author

Pedro Rodrigues (pedror@student.dei.uc.pt)
Alexandre Jesus (ajesus@dei.uc.pt)

Version

0.1

Date

13-09-2021

Copyright

5.4 functor.hpp

Go to the documentation of this file.

```
1
13 #ifndef IBEA_FUNCTOR_HPP
14 #define IBEA_FUNCTOR_HPP
15
16 #include <iostream>
17 #include <random>
18
19 #include "../Utils/solution.hpp"
20 #include "../Utils/wfg.hpp"
21
22 namespace pmnk {
23
29 template <typename R = ObjectiveVector>
30 struct IHD {
31   R m_ref;
32
38   constexpr explicit IHD(R &&ref)
39       : m_ref(std::forward<R>(ref)) {}
40
50   template <typename S = GASolution>
51   [[nodiscard]] double operator()(S const &s1, S const &s2) const {
52     auto const &o1 = s1.objective_vector();
53     auto const &o2 = s2.objective_vector();
54
55     if (weakly_dominates(o1, o2)) {
56       return point_hv(o2, m_ref) - point_hv(o1, m_ref);
57     } else {
58       hvobj<typename R::value_type> hvo(m_ref);
59       hvo.insert(o1), hvo.insert(o2);
60       return hvo.value() - point_hv(o1, m_ref);
61     }
62   }
63 };
64
65
70 struct EPS {
71
76   constexpr explicit EPS() = default;
77
87   template <typename S = GASolution>
88   [[nodiscard]] constexpr double operator()(S const &s1, S const &s2) const noexcept {
89     double indicator = std::numeric_limits<double>::min();
90     for (std::size_t i = 0; i < s1.objective_vector().size(); ++i) {
91       double const o1 = s1.objective_vector()[i];
92       double const o2 = s2.objective_vector()[i];
93       indicator = std::max(indicator, o2 - o1);
94     }
95     return indicator;
96   }
97 };
98
99
105 template <typename RNG>
106 struct NPointCrossover {
107   std::size_t m_crossover_points;
108   double m_crossover_probability;
109   RNG m_rng;
110   std::uniform_real_distribution<double> m_distrib;
111
119   constexpr NPointCrossover(std::size_t const crossover_points, double const crossover_probability,
120                             RNG &rng)
121       : m_crossover_points(crossover_points)
122       , m_crossover_probability(crossover_probability)
123       , m_rng(rng)
124       , m_distrib(0.0, 1.0) {}
125
133   template<typename S = GASolution>
134   void operator()(S &s1, S &s2) noexcept {
135     if (m_distrib(m_rng) < m_crossover_probability) {
136       std::size_t p1 = 0, p2 = 0;
137       for (std::size_t i = 0; i < m_crossover_points; ++i, p1 = p2) {
138         std::uniform_int_distribution<std::size_t> randint(p1, s1.size() - 1);
139         p2 = randint(m_rng);
140         for (auto i = p1; i < p2; ++i) {
141           std::swap(s1[i], s2[i]);
142         };
143       }
144     }
145   }
146 };
147
153 template <typename RNG>
```

```
154 struct UniformCrossover {
155   double m_crossover_probability;
156   RNG m_rng;
157   std::bernoulli_distribution m_distrib;
158 
165   constexpr UniformCrossover(double crossover_probability, RNG &rng)
166       : m_crossover_probability(crossover_probability)
167       , m_rng(rng)
168       , m_distrib() {}
169 
178   template <typename S = GASolution>
179   constexpr void operator()(S &s1, S &s2) noexcept {
180     for (std::size_t i = 0; i < s1.size(); ++i) {
181       if (m_distrib(m_rng)) {
182         std::swap(s1[i], s2[i]);
183       }
184     }
185   }
186 };
187 
193 template <typename RNG>
194 struct UniformMutation {
195   double m_mutation_probability;
196   RNG m_rng;
197   std::uniform_real_distribution<double> m_distrib;
198 
205   constexpr UniformMutation(double const mutation_probability, RNG &rng)
206       : m_mutation_probability(mutation_probability)
207       , m_rng(rng)
208       , m_distrib(0.0, 1.0) {}
209 
217   template <typename S = GASolution>
218   constexpr void operator()(S &s) noexcept {
219     for (std::size_t i = 0; i < s.size(); ++i) {
220       if (m_distrib(m_rng) < m_mutation_probability) {
221         s[i] = !s[i];
222       }
223     }
224   }
225 };
226 
232 template <typename RNG>
233 struct KWayTournamentSelection {
234   std::size_t m_tournament_size;
235   std::size_t m_matting_pool_size;
236   RNG m_rng;
237 
245   constexpr KWayTournamentSelection(std::size_t const tournament_size,
246                                     std::size_t const matting_pool_size, RNG &rng)
247       : m_tournament_size(tournament_size)
248       , m_matting_pool_size(matting_pool_size)
249       , m_rng(rng){};
250 
260   template <typename S = GASolution>
261   [[nodiscard]] std::vector<S> operator()(std::vector<S> const &population) noexcept {
262     std::vector<S> matting_pool;
263     matting_pool.reserve(m_matting_pool_size);
264     std::uniform_int_distribution<std::size_t> distrib(0, population.size() - 1);
265 
266     for (std::size_t i = 0; i < m_matting_pool_size; ++i) {
267       auto best = distrib(m_rng);
268       for (std::size_t j = 0; j < m_tournament_size - 1; ++j) {
269         auto other = distrib(m_rng);
270         best = population[other].fitness() > population[best].fitness() ? other : best;
271       }
272       matting_pool.push_back(population[best]);
273     }
274     return matting_pool;
275   }
276 };
277 
278 }  // namespace pmnk
279 #endif  // IBEA_FUNCTOR_HPP
```

5.5 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/IBEA/ibea.hpp File Reference

IBEA (Indicator Based Evolutionary Algorithm) implementation.

```
#include <csignal>
#include <iomanip>
#include <random>
#include "../Utils/solution.hpp"
#include "../Utils/utils.hpp"
#include "../Utils/wfg.hpp"
#include "functor.hpp"
```

Classes

- class pmnk::IBEA

  Wrapper class for IBEA.

5.5.1 Detailed Description

IBEA (Indicator Based Evolutionary Algorithm) implementation.

Author

Pedro Rodrigues (pedror@student.dei.uc.pt)

Alexandre Jesus (ajesus@dei.uc.pt)

Version

0.1.0

Date

13-09-2021

Copyright

Copyright (c) 2021

5.6 ibea.hpp

Go to the documentation of this file.

```
1
12 #ifndef IBEA_HPP
13 #define IBEA_HPP
14
15 #include <csignal>
16 #include <iomanip>
17 #include <random>
18
19 #include "../Utils/solution.hpp"
20 #include "../Utils/utils.hpp"
21 #include "../Utils/wfg.hpp"
22 #include "functor.hpp"
23
24 namespace pmnk {
25
27 class IBEA {
28  public:
29   RMNKEval eval;
```

```
30
31  private:
32   std::mt19937 m_generator;
33   std::ostream &m_os;
34
35   hvobj<ObjectiveVector::value_type> m_hvo;
36   std::vector<GASolution> m_solutions;
37
38  public:
50   template <typename Str = std::string, typename Ref = ObjectiveVector>
51   IBEA(Str &&instance, unsigned int const seed, std::ostream &os, Ref &&ref)
52       : eval(std::forward<Str>(instance).c_str())
53       , m_generator(seed)
54       , m_os(os)
55       , m_hvo(std::forward<Ref>(ref)) {}
56
66   template <typename Str = std::string>
67   IBEA(Str &&instance, unsigned int const seed, std::ostream &os)
68       : eval(std::forward<Str>(instance).c_str())
69       , m_generator(seed)
70       , m_os(os)
71       , m_hvo(ObjectiveVector(eval.getM(), 0.0)) {}
72
83   template <typename Str = std::string, typename Ref = ObjectiveVector>
84   IBEA(Str &&instance, unsigned int const seed, Ref &&ref)
85       : IBEA(std::forward<Str>(instance), seed, std::cout, std::forward<Ref>(ref)) {}
86
95   template <typename Str = std::string>
96   IBEA(Str &&instance, unsigned int const seed)
97       : IBEA(std::forward<Str>(instance), seed, std::cout) {}
98
109   template <typename Str = std::string, typename Ref = ObjectiveVector>
110   explicit IBEA(Str &&instance, Ref &&ref)
111       : IBEA(std::forward<Str>(instance), std::random_device()(), std::cout,
112              std::forward<Ref>(ref)) {}
113
121   template <typename Str = std::string>
122   explicit IBEA(Str &&instance)
123       : IBEA(std::forward<Str>(instance), std::random_device()(), std::cout) {}
124
131   std::vector<GASolution> const &solutions() const {
132     return m_solutions;
133   }
134
154   template <typename I, typename S, typename M, typename C>
155   void run(std::size_t const maxeval, std::size_t const population_max_size,
156            std::size_t const max_generations, double const scaling_factor, I &&indicator,
157            C &&crossover_method, M &&mutation_method, S &&selection_method, bool adaptive) {
158     if (adaptive) {
159       m_run<true>(maxeval, population_max_size, max_generations, scaling_factor, indicator,
160                   crossover_method, mutation_method, selection_method);
161     } else {
162       m_run<false>(maxeval, population_max_size, max_generations, scaling_factor, indicator,
163                    crossover_method, mutation_method, selection_method);
164     }
165   }
166
167  private:
187   template <bool Adaptive, typename I, typename S, typename M, typename C>
188   void m_run(std::size_t const maxeval, std::size_t const pop_max,
189              std::size_t const max_generations, double const scaling_factor, I &&indicator,
190              C &&crossover_method, M &&mutation_method, S &&selection_method) {
191     std::size_t evaluation = 0, gen = 0;
192     double c = 1;
193
194     std::vector<GASolution> population;
195     population.reserve(pop_max);
196
197     std::cout « "evaluation,generation,hypervolume\n";
198
199     for (std::size_t i = 0; i < pop_max && evaluation < maxeval; ++i) {
200       auto sol = GASolution(Solution::random_solution(eval, m_generator));
201       if (add_non_dominated(m_solutions, sol)) {
202         m_hvo.insert(sol.objective_vector());
203         m_os « std::setprecision(12) « evaluation « "," « gen « "," « m_hvo.value() « "\n";
204       }
205       population.push_back(std::move(sol));
206       ++evaluation;
207     }
208
209     if (evaluation < maxeval) {
210       if constexpr (Adaptive) {
211         c = m_adaptive_factor(population, indicator);
212       }
213       m_fitness_assignment(population, scaling_factor * c, indicator);
214     }
215
```

```
216     for (; evaluation < maxeval && gen < max_generations; ++gen) {
217       auto matting_pool = selection_method(population);
218
219       for (std::size_t i = 0; i < matting_pool.size() - 1; i += 2) {
220         crossover_method(matting_pool[i], matting_pool[i + 1]);
221       }
222
223       for (auto &individual : matting_pool) {
224         mutation_method(individual);
225         individual.eval(eval);
226       }
227
228       if constexpr (Adaptive) {
229         c = m_adaptive_factor(population, indicator);
230       }
231       m_fitness_assignment(population, scaling_factor * c, indicator);
232
233       for (auto &individual : matting_pool) {
234         if (add_non_dominated(m_solutions, individual)) {
235           m_hvo.insert(individual.objective_vector());
236           m_os « std::setprecision(12) « evaluation « "," « gen « "," « m_hvo.value() « "\n";
237         }
238         population.push_back(std::move(individual));
239         ++evaluation;
240       }
241       m_environmental_selection(population, scaling_factor * c, pop_max, indicator);
242     }
243     m_os « std::setprecision(12) « evaluation « "," « gen « "," « m_hvo.value() « "\n";
244   }
245
254   template <typename S = GASolution>
255   auto m_objective_bounds(std::vector<S> const &population) {
256     auto ub = std::numeric_limits<ObjectiveVector::value_type>::min();
257     auto lb = std::numeric_limits<ObjectiveVector::value_type>::max();
258     for (auto const &individual : population) {
259       for (auto const v : individual.objective_vector()) {
260         lb = std::min(lb, v);
261         ub = std::max(ub, v);
262       }
263     }
264     return std::make_pair(lb, ub);
265   }
266
277   template <typename S = GASolution>
278   auto m_scale_objective_vectors(std::vector<S> const &population, double const lb,
279                                  double const ub) {
280     std::vector<S> s = population;
281     for (auto &individual : s) {
282       auto ov = individual.objective_vector();
283       for (auto &i : ov) {
284         i = (i - ub) / (ub - lb);
285       }
286       individual.set_objv(std::move(ov));
287     }
288     return s;
289   }
290
300   template <typename I, typename S = GASolution>
301   auto m_adaptive_factor(std::vector<S> const &population, I &&indicator) {
302     auto &&[lb, ub] = m_objective_bounds(population);
303     auto s = m_scale_objective_vectors(population, lb, ub);
304     auto c = std::numeric_limits<ObjectiveVector::value_type>::min();
305     for (std::size_t i = 0; i < s.size(); ++i) {
306       for (std::size_t j = 0; j < s.size(); ++j) {
307         if (i != j) {
308           c = std::max(c, std::abs(indicator(s[i], s[j])));
309         }
310       }
311     }
312     return c;
313   }
314
324   template <typename I, typename S = GASolution>
325   void m_fitness_assignment(std::vector<S> &population, double const k, I &&indicator) const {
326     for (std::size_t i = 0; i < population.size(); ++i) {
327       population[i].set_fitness(0);
328       for (std::size_t j = 0; j < population.size(); ++j) {
329         if (i != j) {
330           population[i].set_fitness(population[i].fitness() -
331                                     std::exp(-indicator(population[j], population[i]) / k));
332         }
333       }
334     }
335   }
336
349   template <typename I, typename S = GASolution>
350   void m_environmental_selection(std::vector<S> &population, double const k,
```

```
351                                 std::size_t population_max_size, I &&indicator) {
352     while (population.size() > population_max_size) {
353       std::size_t worst = 0;
354       for (std::size_t i = 0; i < population.size(); ++i) {
355         worst = population[i].fitness() < population[worst].fitness() ? i : worst;
356       }
357
358       std::swap(population[worst], population.back());
359       for (std::size_t i = 0; i < population.size() - 1; ++i) {
360         population[i].set_fitness(population[i].fitness() +
361                                   std::exp(-indicator(population.back(), population[i]) / k));
362       }
363       population.pop_back();
364     }
365   }
366 };
367 }  // namespace pmnk
368 #endif  // IBEA_HPP
```

5.7 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/main.cpp File Reference

Driver program to test the implementation of some search algorithms and simplify the gathering of anytime data relevant to the study of their performance in the context of the pmnk-landscapes problem.

```
#include <CLI/CLI.hpp>
#include <fstream>
#include <iostream>
#include <random>
#include <tuple>
#include "GSEMO/gsemo.hpp"
#include "IBEA/ibea.hpp"
#include "PLS/pls.hpp"
```

Macros

- #define SELECTION(MAXEVAL, POP, GEN, FACTOR, I, C, M, S, ADAPT)
- #define MUTATION(MAXEVAL, POP, GEN, FACTOR, I, C, M, S, ADAPT)
- #define CROSSOVER(MAXEVAL, POP, GEN, FACTOR, I, C, M, S, ADAPT)
- #define INDICATOR(MAXEVAL, POP, GEN, FACTOR, I, C, M, S, ADAPT)
- #define **RUN_IBEA_LOOP**(...) INDICATOR(__VA_ARGS__)

Enumerations

- enum class Indicator { **EPS** , **IHD** }

  Enum representing which indicator to use as the IBEA algorithm indicator operator. Possible values are:
- enum class Crossover { **NPC** , **UC** }

  Enum representing which indicator to use as the IBEA algorithm crossover operator. Possible values are:
- enum class Selection { **KWT** }

  Enum representing which indicator to use as the IBEA algorithm selection operator. Possible values are:
- enum class Mutation { **UM** }

  Enum representing which indicator to use as the IBEA algorithm selection operator. Possible values are:

Functions

- void set_positional_arguments (CLI::App &app, std::string &instance)

  Set the CLI positional arguments options/flags.
- void set_general_options (CLI::App &app, std::size_t &maxeval, unsigned int &seed, std::string &output, ObjectiveVector &ref)

  Set the CLI general options/flags.
- void set_pls_options (CLI::App &app, PLSAcceptanceCriterion &pac, PLSExplorationCriterion &pne)

  Set the PLS algorithm options/flags.
- void set_ibea_options (CLI::App &app, std::size_t &population_size, std::size_t &generations, double &scaling_factor, bool &adaptive)

  Set the IBEA algorithm options/flags.
- void set_ibea_mutation_options (CLI::App &app, double &mutation_probability)

  Set the IBEA algorithm mutation operators options/flags.
- void set_ibea_crossover_options (CLI::App &app, double &crossover_probability, std::size_t &npoints)

  Set the IBEA algorithm crossover operators options/flags.
- void set_ibea_selection_options (CLI::App &app, std::size_t &matting_pool_size, std::size_t &tournament_↩ size)

  Set the ibea selection options object.
- void gsemo (std::string const &instance, std::size_t const maxeval, unsigned int const seed, std::ostream &os, ObjectiveVector &ref)

  CLI::App callback for the gsemo algorithm.
- void pls (std::string const &instance, std::size_t maxeval, unsigned int seed, PLSAcceptanceCriterion const pac, PLSExplorationCriterion const pne, std::ostream &os, ObjectiveVector &ref)

  CLI::App callback for the pls algorithm.
- void ibea (std::string const &instance, std::size_t const maxeval, unsigned int const seed, std::size_t const ps, std::size_t const gen, double const k, double const mp, double const cp, std::size_t npts, std::size_t const mps, std::size_t const ts, Indicator const indicator, Crossover const crossover, Mutation const mutation, Selection const selection, bool adaptive, std::ostream &os, ObjectiveVector &ref)

  CLI::App callback for the ibea algorithm.
- int **main** (int argc, char ∗∗argv)

5.7.1 Detailed Description

Driver program to test the implementation of some search algorithms and simplify the gathering of anytime data relevant to the study of their performance in the context of the pmnk-landscapes problem.

Author

Pedro Rodrigues (pedror@student.dei.uc.pt)

Alexandre Jesus (ajesus@dei.uc.pt)

Version

0.1.0

Date

13-09-2021

Copyright

5.7.2 Macro Definition Documentation

5.7.2.1 CROSSOVER

```
#define CROSSOVER(
              MAXEVAL,
              POP,
              GEN,
              FACTOR,
              I,
              C,
              M,
              S,
              ADAPT )
```

Value:

```
  switch (C) {                                                                          \
    case Crossover::NPC:                                                                \
      MUTATION(MAXEVAL, POP, GEN, FACTOR, I, NPointCrossover(npts, cp, rng), M, S, ADAPT) \
      break;                                                                            \
    case Crossover::UC:                                                                 \
      MUTATION(MAXEVAL, POP, GEN, FACTOR, I, UniformCrossover(cp, rng), M, S, ADAPT)    \
      break;                                                                            \
    default:                                                                            \
      throw("Unknown crossover operator!\n");                                           \
  }
```

5.7.2.2 INDICATOR

```
#define INDICATOR(
              MAXEVAL,
              POP,
              GEN,
              FACTOR,
              I,
              C,
              M,
              S,
              ADAPT )
```

Value:

```
  switch (I) {                                                                          \
    case Indicator::EPS:                                                                \
      CROSSOVER(MAXEVAL, POP, GEN, FACTOR, EPS(), C, M, S, ADAPT)                       \
      break;                                                                            \
    case Indicator::IHD:                                                                \
      CROSSOVER(MAXEVAL, POP, GEN, FACTOR, IHD(ObjectiveVector(ibea.eval.getM(), 0)), C, M, S, \
                ADAPT)                                                                  \
      break;                                                                            \
    default:                                                                            \
      throw("Unknown indicator!\n");                                                    \
  }
```

5.7.2.3 MUTATION

```
#define MUTATION(
                MAXEVAL,
                POP,
                GEN,
                FACTOR,
                I,
                C,
                M,
                S,
                ADAPT )
```

Value:

```
  switch (M) {                                                                    \
    case Mutation::UM:                                                            \
      SELECTION(MAXEVAL, POP, GEN, FACTOR, I, C, UniformMutation(mp, rng), S, ADAPT) \
      break;                                                                      \
    default:                                                                      \
      throw("Unknown mutation operator!\n");                                      \
  }
```

5.7.2.4 SELECTION

```
#define SELECTION(
                MAXEVAL,
                POP,
                GEN,
                FACTOR,
                I,
                C,
                M,
                S,
                ADAPT )
```

Value:

```
  switch (S) {                                                                    \
    case Selection::KWT:                                                          \
      if (ref.empty()) {                                                          \
        IBEA ibea(instance, seed, os);                                            \
        ibea.run(MAXEVAL, POP, GEN, FACTOR, I, C, M, KWayTournamentSelection(ts, mps, rng), \
                 ADAPT);                                                          \
      } else {                                                                    \
        IBEA ibea(instance, seed, os, ref);                                       \
        ibea.run(MAXEVAL, POP, GEN, FACTOR, I, C, M, KWayTournamentSelection(ts, mps, rng), \
                 ADAPT);                                                          \
      }                                                                           \
      break;                                                                      \
    default:                                                                      \
      throw("Unknown selection method!");                                         \
  }
```

5.7.3 Enumeration Type Documentation

5.7.3.1 Crossover

```
enum class Crossover  [strong]
```

Enum representing which indicator to use as the IBEA algorithm crossover operator. Possible values are:

- NPC (N-Point Crossover)
- UC (Uniform Crossover)

5.7.3.2 Indicator

```
enum class Indicator  [strong]
```

Enum representing which indicator to use as the IBEA algorithm indicator operator. Possible values are:

- EPS (Additive Epsilon Indicator)
- IHD (Hypervolume Based Indicator)

5.7.3.3 Mutation

```
enum class Mutation  [strong]
```

Enum representing which indicator to use as the IBEA algorithm selection operator. Possible values are:

- UM (Uniform Mutation)

5.7.3.4 Selection

```
enum class Selection  [strong]
```

Enum representing which indicator to use as the IBEA algorithm selection operator. Possible values are:

- KWT (K-Way Tournament Selection)

5.7.4 Function Documentation

5.7.4.1 gsemo()

```
void gsemo (
            std::string const & instance,
            std::size_t const maxeval,
            unsigned int const seed,
            std::ostream & os,
            ObjectiveVector & ref )  [inline]
```

CLI::App callback for the gsemo algorithm.

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
maxeval	The maximum number of evaluations performed by the algorithms (stopping criterion)
seed	The seed used by the pseudo random number generator used in these algorithms
os	The name of the output file where the standard output stream should be redirected
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

5.7.4.2 ibea()

```
void ibea (
            std::string const & instance,
            std::size_t const maxeval,
            unsigned int const seed,
            std::size_t const ps,
            std::size_t const gen,
            double const k,
            double const mp,
            double const cp,
            std::size_t npts,
            std::size_t const mps,
            std::size_t const ts,
            Indicator const indicator,
            Crossover const crossover,
            Mutation const mutation,
            Selection const selection,
            bool adaptive,
            std::ostream & os,
            ObjectiveVector & ref )  [inline]
```

CLI::App callback for the ibea algorithm.

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
maxeval	The maximum number of evaluations performed by the algorithms (stopping criterion)
seed	The seed used by the pseudo random number generator used in these algorithms
ps	The maximum population size
gen	The maximum number of generations
k	The scaling factor
mp	The mutation probability considered by the mutation operator
cp	The crossover probability considered by the crossover operator
npts	The number of crossover points considered by the NPointCrossover operator if used
mps	The matting pool size
ts	The size of the tournament considered by the KWayTournament operator
indicator	The indicator to be used by the IBEA indicator operator
crossover	The crossover method considered by the IBEA crossover operator

Parameters

mutation	The mutation method considered by the IBEA mutation operator
selection	The selection method considered by the IBEA selection operator
adaptive	boolean indicative of version of IBEA to be used. If true use adaptive version of (A-IBEA) else use (B-IBEA)
os	The name of the output file where the standard output stream should be redirected
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

Helper define to avoid the use of runtime polymorphism methods to distinguish between selection operators that ibea is going to use during its execution

Helper define to avoid the use of runtime polymorphism methods to distinguish between crossover operators that ibea is going to use during its execution

Helper define to avoid the use of runtime polymorphism methods to distinguish between crossover operators that ibea is going to use during its execution.

\ Helper define to avoid the use of runtime polymorphism methods to distinguish between indicator operators that ibea is going to use during its execution.

Helper define that forwards all the information to the others

5.7.4.3 pls()

```
void pls (
            std::string const & instance,
            std::size_t maxeval,
            unsigned int seed,
            PLSAcceptanceCriterion const pac,
            PLSExplorationCriterion const pne,
            std::ostream & os,
            ObjectiveVector & ref )  [inline]
```

CLI::App callback for the pls algorithm.

Parameters

instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script
maxeval	The maximum number of evaluations performed by the algorithms (stopping criterion)
seed	The seed used by the pseudo random number generator used in these algorithms
pac	The PLS algorithm solution acceptance criterion
pne	The PLS algorithm neighboorhood solutions exploration criterion
os	The name of the output file where the standard output stream should be redirected
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

5.7.4.4 set_general_options()

```
void set_general_options (
            CLI::App & app,
            std::size_t & maxeval,
            unsigned int & seed,
            std::string & output,
            ObjectiveVector & ref )  [inline]
```

Set the CLI general options/flags.

Parameters

app	CLI::App object that will hold all the global options/flags (below).
maxeval	The maximum number of evaluations performed by the algorithms (stopping criterion)
seed	The seed used by the pseudo random number generator used in these algorithms
output	The name of the output file where the standard output stream should be redirected
ref	The reference point considered by hypervolume indicator whilst running the algorithms (anytime measure)

5.7.4.5 set_ibea_crossover_options()

```
void set_ibea_crossover_options (
            CLI::App & app,
            double & crossover_probability,
            std::size_t & npoints )  [inline]
```

Set the IBEA algorithm crossover operators options/flags.

Parameters

app	CLI::App object that will hold all the IBEA crossover operator options/flags (below).
crossover_probability	The probability of occurrence of crossover between two individual's genotypes
npoints	If the crossover operator is the n-point crossover this option is set. This option represents the number of crossover points to be considered when swapping both individual's genetic content

5.7.4.6 set_ibea_mutation_options()

```
void set_ibea_mutation_options (
            CLI::App & app,
            double & mutation_probability )  [inline]
```

Set the IBEA algorithm mutation operators options/flags.

Parameters

app	CLI::App object that will hold all the IBEA mutation operator options/flags (below).
mutation_probability	The probability of occurrence of mutations in the individual's genotype

5.7.4.7 set_ibea_options()

```
void set_ibea_options (
            CLI::App & app,
            std::size_t & population_size,
            std::size_t & generations,
            double & scaling_factor,
            bool & adaptive )  [inline]
```

Set the IBEA algorithm options/flags.

Parameters

app	CLI::App object that will hold all the IBEA options/flags (below).
population_size	The maximum size the population
generations	The maximum number of generations (stopping criterion)
scaling_factor	The IBEA scaling factor
adaptive	A boolean indicative of the version of the algorithm to be used. True for B-IBEA (Basic IBEA) and false for A-IBEA (Adaptive IBEA)

5.7.4.8 set_ibea_selection_options()

```
void set_ibea_selection_options (
            CLI::App & app,
            std::size_t & matting_pool_size,
            std::size_t & tournament_size )  [inline]
```

Set the ibea selection options object.

Parameters

app	CLI::App object that will hold all the IBEA selection operator options/flags (below).
matting_pool_size	The size of the matting pool.
tournament_size	The size of the tournament used for selection.

5.7.4.9 set_pls_options()

```
void set_pls_options (
```

```
            CLI::App & app,
            PLSAcceptanceCriterion & pac,
            PLSExplorationCriterion & pne )  [inline]
```

Set the PLS algorithm options/flags.

Parameters

app	CLI::App object that will hold all the PLS options/flags (below).
pac	The PLS algorithm solution acceptance criterion
pne	The PLS algorithm neighboorhood solutions exploration criterion

5.7.4.10 set_positional_arguments()

```
void set_positional_arguments (
            CLI::App & app,
            std::string & instance )  [inline]
```

Set the CLI positional arguments options/flags.

Parameters

app	CLI::App object that will hold all the options/flags of the positional arguments (below).
instance	The path for the "rmnk" instance (.dat) file to be used. These files can be generated using the rmnkGenerator.R script

5.8 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/PLS/pls.hpp File Reference

PLS (Pareto Local Search) algorithm implementation.

```
#include <csignal>
#include <iomanip>
#include "../Utils/solution.hpp"
#include "../Utils/utils.hpp"
#include "../Utils/wfg.hpp"
```

Classes

- class pmnk::PLS

  Wrapper class for PLS.

Macros

- #define RUNLOOP(FIRSTIMPROV)

Enumerations

- enum class **PLSAcceptanceCriterion** { **NON_DOMINATING** , **DOMINATING** , **BOTH** }
- enum class **PLSExplorationCriterion** { **BEST_IMPROVEMENT** , **FIRST_IMPROVEMENT** , **BOTH** }

5.8.1 Detailed Description

PLS (Pareto Local Search) algorithm implementation.

Author

Pedro Rodrigues (pedror@student.dei.uc.pt)

Alexandre Jesus (ajesus@dei.uc.pt)

Version

0.1.0

Date

13-09-2021

Copyright

Copyright (c) 2021

5.8.2 Macro Definition Documentation

5.8.2.1 RUNLOOP

```
#define RUNLOOP(
              FIRSTIMPROV )
```

Value:

```
  switch (acceptance_criterion) {                                                \
    case PLSAcceptanceCriterion::NON_DOMINATING:                                 \
      m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::NON_DOMINATING>(eval, maxeval); \
      break;                                                                     \
    case PLSAcceptanceCriterion::DOMINATING:                                     \
      m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::DOMINATING>(eval, maxeval);    \
      break;                                                                     \
    case PLSAcceptanceCriterion::BOTH:                                           \
      m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::BOTH>(eval, maxeval);          \
      break;                                                                     \
    default:                                                                     \
      throw("Unknown value for acceptance criterion");                           \
  }
```

5.9 pls.hpp

Go to the documentation of this file.

```
1
13 #ifndef PLS_HPP
14 #define PLS_HPP
15
16 #include <csignal>
17 #include <iomanip>
18
19 #include "../Utils/solution.hpp"
20 #include "../Utils/utils.hpp"
21 #include "../Utils/wfg.hpp"
22
23 namespace pmnk {
24
25 // Acceptance criterion:
26 //   - 0 -> accept every non-dominated neighbor (NON_DOMINATING)
27 //   - 1 -> accept only neighbors that dominate current solution (DOMINATING)
28 //   - 2 -> first try to accept only neighbors that dominate the (BOTH)
29 //          current solution, if none exist accept non-dominated
30
31 enum class PLSAcceptanceCriterion { NON_DOMINATING, DOMINATING, BOTH };
32
33 // Neighborhood exploration:
34 //   - 0 -> explore every acceptable neighbor (BEST_IMPROVEMENT)
35 //   - 1 -> stop once one neighbor is accepted (FIRST_IMPROVEMENT)
36 //   - 2 -> use 1 until PLS stops, afterwards restart and use 0 (BEST_IMPROVEMENT)
37 enum class PLSExplorationCriterion { BEST_IMPROVEMENT, FIRST_IMPROVEMENT, BOTH };
38
40 class PLS {
41  public:
42   RMNKEval eval;
43
44  private:
45   std::mt19937 m_generator;
46   std::ostream &m_os;
47
48   hvobj<typename ObjectiveVector::value_type> m_hvo;
49   std::vector<Solution> m_solutions;
50   std::vector<Solution> m_non_visited_solutions;
51
52  public:
53
65   template <typename Str = std::string, typename Ref = ObjectiveVector>
66   PLS(Str &&instance, unsigned int seed, std::ostream &os, Ref &&ref)
67       : eval(std::forward<Str>(instance).c_str())
68       , m_generator(seed)
69       , m_os(os)
70       , m_hvo(std::forward<Ref>(ref)) {}
71
81   template <typename Str = std::string>
82   PLS(Str &&instance, unsigned int seed, std::ostream &os)
83       : eval(std::forward<Str>(instance).c_str())
84       , m_generator(seed)
85       , m_os(os)
86       , m_hvo(ObjectiveVector(eval.getM(), 0.0)) {}
87
98   template <typename Str = std::string, typename Ref = ObjectiveVector>
99   PLS(Str &&instance, unsigned int seed, Ref &&ref)
100       : PLS(std::forward<Str>(instance), seed, std::cout, std::forward<Ref>(ref)) {}
101
110   template <typename Str = std::string>
111   PLS(Str &&instance, unsigned int seed)
112       : PLS(std::forward<Str>(instance), seed, std::cout) {}
113
123   template <typename Str = std::string, typename Ref = ObjectiveVector>
124   PLS(Str &&instance, Ref &&ref)
125       : PLS(std::forward<Str>(instance), std::random_device()(), std::cout,
126             std::forward<Ref>(ref)) {}
127
135   template <typename Str = std::string>
136   PLS(Str &&instance)
137       : PLS(std::forward<Str>(instance), std::random_device()(), std::cout) {}
138
145   std::vector<Solution> const solutions() const {
146     return m_solutions;
147   }
148
156   std::vector<Solution> const non_visited_solutions() const {
157     return m_non_visited_solutions;
158   }
159
168   void run(std::size_t maxeval, PLSAcceptanceCriterion const acceptance_criterion,
169            PLSExplorationCriterion const neighborhood_exploration) {
```

```
170     auto rand_solution = Solution::random_solution(eval, m_generator);
171     m_hvo.insert(rand_solution.objective_vector());
172
173     add_non_dominated(m_non_visited_solutions, std::move(rand_solution));
174     m_solutions = m_non_visited_solutions;
175
176     std::size_t eval = 0;
177     m_os « "evaluation,hypervolume\n";
178     m_os « std::setprecision(12) « eval « "," « m_hvo.value() « "\n";
179
180 #define RUNLOOP(FIRSTIMPROV)                                                    \
181   switch (acceptance_criterion) {                                               \
182     case PLSAcceptanceCriterion::NON_DOMINATING:                                \
183       m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::NON_DOMINATING>(eval, maxeval); \
184       break;                                                                    \
185     case PLSAcceptanceCriterion::DOMINATING:                                    \
186       m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::DOMINATING>(eval, maxeval);   \
187       break;                                                                    \
188     case PLSAcceptanceCriterion::BOTH:                                          \
189       m_loop<FIRSTIMPROV, PLSAcceptanceCriterion::BOTH>(eval, maxeval);         \
190       break;                                                                    \
191     default:                                                                    \
192       throw("Unknown value for acceptance criterion");                          \
193   }
194
195     if (neighborhood_exploration == PLSExplorationCriterion::BEST_IMPROVEMENT) {
196       RUNLOOP(false);
197     } else if (neighborhood_exploration == PLSExplorationCriterion::FIRST_IMPROVEMENT) {
198       RUNLOOP(true);
199     } else if (neighborhood_exploration == PLSExplorationCriterion::BOTH) {
200       RUNLOOP(true);
201       RUNLOOP(false);
202     } else {
203       throw("Unknown value for neighborhood exploration");
204     }
205   }
206
207  private:
208
220   template <bool FirstImprov, PLSAcceptanceCriterion Acceptance>
221   void m_loop(size_t &evaluation, size_t maxeval) {
222     while (evaluation < maxeval && !m_non_visited_solutions.empty()) {
223       std::uniform_int_distribution<std::size_t> distrib(0, m_non_visited_solutions.size() - 1);
224       std::size_t index = distrib(m_generator);
225
226       auto original = std::move(m_non_visited_solutions[index]);
227       m_non_visited_solutions[index] = std::move(m_non_visited_solutions.back());
228       m_non_visited_solutions.pop_back();
229
230       if constexpr (Acceptance == PLSAcceptanceCriterion::NON_DOMINATING) {
231         for (size_t i = 0; i < original.decision_vector().size() && evaluation < maxeval; ++i) {
232           DecisionVector decision_vector = original.decision_vector();
233           decision_vector[i] = !decision_vector[i];
234           auto solution = Solution(eval, std::move(decision_vector));
235           ++evaluation;
236           if (add_non_dominated(m_solutions, solution)) {
237             m_hvo.insert(solution.objective_vector());
238             add_non_dominated(m_non_visited_solutions, std::move(solution));
239             m_os « std::setprecision(12) « evaluation « "," « m_hvo.value() « "\n";
240             if constexpr (FirstImprov) {
241               break;
242             }
243           }
244         }
245       } else if constexpr (Acceptance == PLSAcceptanceCriterion::DOMINATING) {
246         for (size_t i = 0; i < original.decision_vector().size() && evaluation < maxeval; ++i) {
247           DecisionVector decision_vector = original.decision_vector();
248           decision_vector[i] = !decision_vector[i];
249           auto solution = Solution(eval, std::move(decision_vector));
250           ++evaluation;
251           if (solution.dominance(original) == DominanceType::DOMINATES &&
252               add_non_dominated(m_solutions, solution)) {
253             m_hvo.insert(solution.objective_vector());
254             add_non_dominated(m_non_visited_solutions, std::move(solution));
255             m_os « std::setprecision(12) « evaluation « "," « m_hvo.value() « "\n";
256             if constexpr (FirstImprov) {
257               break;
258             }
259           }
260         }
261       } else if constexpr (Acceptance == PLSAcceptanceCriterion::BOTH) {
262         std::vector<std::pair<Solution, size_t» remaining;
263         remaining.reserve(original.decision_vector().size());
264         bool use_remaining = true;
265         for (size_t i = 0; i < original.decision_vector().size() && evaluation < maxeval; ++i) {
266           DecisionVector decision_vector = original.decision_vector();
267           decision_vector[i] = !decision_vector[i];
```

```
268           auto solution = Solution(eval, std::move(decision_vector));
269           ++evaluation;
270           if (solution.dominance(original) == DominanceType::DOMINATES &&
271               add_non_dominated(m_solutions, solution)) {
272             use_remaining = false;
273             m_hvo.insert(solution.objective_vector());
274             add_non_dominated(m_non_visited_solutions, std::move(solution));
275             m_os « std::setprecision(12) « evaluation « "," « m_hvo.value() « "\n";
276             if constexpr (FirstImprov) {
277               break;
278             }
279           } else if (use_remaining) {
280             remaining.emplace_back(std::move(solution), evaluation);
281           }
282         }
283         if (use_remaining) {
284           for (auto &&[solution, iteration] : remaining) {
285             if (add_non_dominated(m_solutions, solution)) {
286               m_hvo.insert(solution.objective_vector());
287               add_non_dominated(m_non_visited_solutions, std::move(solution));
288               m_os « std::setprecision(12) « iteration « "," « m_hvo.value() « "\n";
289               if constexpr (FirstImprov) {
290                 break;
291               }
292             }
293           }
294         }
295       }
296     }
297   }
298 };
299 }  // namespace pmnk
300 #endif  // PLS_HPP
```

5.10 rMNKEval.hpp

```
1 /*
2     This library is free software; you can redistribute it and/or
3     modify it under the terms of the GNU Lesser General Public
4     License as published by the Free Software Foundation; version 3
5     of the License.
6
7     This library is distributed in the hope that it will be useful,
8     but WITHOUT ANY WARRANTY; without even the implied warranty of
9     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
10     Lesser General Public License for more details.
11
12     You should have received a copy of the GNU Lesser General Public
13     License along with this library; if not, write to the Free Software
14     Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
15
16 Contact: http://mocobench.sourceforge.net
17
18 Authors:
19     Arnaud Liefooghe <arnaud.liefooghe@lifl.fr>
20     Sebastien Verel  <sebastien.verel@inria.fr>
21 */
22
23 #ifndef __rMNKEval
24 #define __rMNKEval
25
26 /*
27  *  Fitness function of the rMNK-landscapes
28  *  reading a rhoMNK-landscapes instance file
29  *  in c++ style
30  *
31  *  rhoMNK-landscapes instances can be generated with the rmnkGenerator.R
32  *
33  *  More information on rhoMNK-landscapes, see original paper:
34  *  Verel S., Liefooghe A., Jourdan L., Dhaenens C. "Analyzing the Effect of Objective Correlation
35  * on the Efficient Set of MNK-Landscapes", In Proceedings of Learning and Intelligent OptimizatioN
36  * Conference (LION 5), LNCS, p. , 2011.
37  */
38
39 #include <fstream>
40 #include <iostream>
41 #include <string>
42 #include <vector>
43
44 namespace pmnk {
45
47 class RMNKEval {
48  public:
```

```
49   /*
50    *  Constructor
51    *
52    * @param _fileName file name instance of the rho MNK-landscapes
53    */
54   RMNKEval(const char *_fileName) {
55     load(_fileName);
56   }
57
58   /*
59    *  Destructor
60    */
61   ~RMNKEval() {
62     if (links != NULL) {
63       for (unsigned int n = 0; n < M; n++) {
64         for (unsigned int i = 0; i < N; i++)
65           delete[](links[n][i]);
66         delete[] links[n];
67       }
68
69       delete[] links;
70       links = NULL;
71     }
72
73     if (tables != NULL) {
74       for (unsigned int n = 0; n < M; n++) {
75         for (unsigned int i = 0; i < N; i++)
76           delete[](tables[n][i]);
77         delete[] tables[n];
78       }
79
80       delete tables;
81       tables = NULL;
82     }
83   }
84
85   /*
86    * Compute the fitness function
87    *
88    * @param _solution the solution to evaluate
89    * @param _objVec   the objective vector of the corresponding solution
90    */
91   void eval(std::vector<bool> &_solution, std::vector<double> &_objVec) {
92     _objVec.resize(M);
93
94     for (unsigned n = 0; n < M; n++)
95       _objVec[n] = evalNK(n, _solution);
96   }
97
98   /*
99    * to get objective space dimension
100   *
101   * @return dimension of the objective space
102   */
103  unsigned getM() {
104    return M;
105  }
106
107  /*
108   * to get bitstring size
109   *
110   * @return dimension of the bitstring
111   */
112  unsigned getN() {
113    return N;
114  }
115
116  /*
117   * to get epistasis degree (K)
118   *
119   * @return epistasis degree K
120   */
121  unsigned getK() {
122    return K;
123  }
124
125  /*
126   * to get the correlation between each tuple of contributions
127   *
128   * @return parameter rho
129   */
130  double getRho() {
131    return rho;
132  }
133
134  protected:
135   // correlation between contributions
```

```
136   double rho;
137
138   // number of objective functions
139   unsigned M;
140
141   // size of the bit string
142   unsigned N;
143
144   // number of interactions between variables (epistasis)
145   unsigned K;
146
147   // the M tables of contributions
148   double ***tables;
149
150   // the M links description
151   unsigned ***links;
152
153   /**********************************************
154    *
155    * Load the file of a rMNK-landscapes instance
156    *
157    * @param fileName file name instance of the rMNK-landscapes
158    *
159    **********************************************/
160   virtual void load(const char *_fileName) {
161     std::fstream file;
162     file.open(_fileName, std::ios::in);
163
164     if (file.is_open()) {
165       std::string s;
166
167       // read the commentaries
168       std::string line;
169
170       file » s;
171       while (s[0] == 'c') {
172         getline(file, line, '\n');
173         file » s;
174       }
175
176       // read the parameters
177       if (s.compare("p") != 0)
178         std::cerr « "Error RMNKEval.load: expected line beging by \"p\" at " + s + " in "
179                  « _fileName;
180
181       file » s;
182
183       if (s.compare("rMNK") != 0)
184         std::cerr « "Error RMNKEval.load: type rMNK expected at " + s + " in " « _fileName;
185
186       // effective read of the parameters
187       file » rho » M » N » K;
188
189       init();
190
191       file » s;
192       // read the links
193       if (s.compare("p") != 0)
194         std::cerr « "Error RMNKEval.load: expected line beging by \"p\" at " + s + " in "
195                  « _fileName;
196
197       file » s;
198       if (s.compare("links") == 0)
199         loadLinks(file);
200       else
201         std::cerr « "Error RMNKEval.load: line with \"links\" expected at " + s + " in "
202                  « _fileName;
203
204       // read the tables of contributions
205       file » s;
206       if (s.compare("p") != 0)
207         std::cerr « "Error RMNKEval.load: expected line beging by \"p\" at " + s + " in "
208                  « _fileName;
209
210       file » s;
211
212       if (s.compare("tables") == 0)
213         loadTables(file);
214       else
215         std::cerr « "Error RMNKEval.load: line with \"tables\" expected at " + s + " in "
216                  « _fileName;
217
218       file.close();
219     } else
220       std::cerr « "Error RMNKEval.load: impossible to open file " « _fileName;
221   };
222
```

```
223   /*********************************************
224    *
225    * Initialization of the different tables and epistasis links
226    *
227    *********************************************/
228   void init() {
229     links = new unsigned **[M];
230     tables = new double **[M];
231
232     for (unsigned n = 0; n < M; n++) {
233       links[n] = new unsigned *[N];
234       tables[n] = new double *[N];
235
236       for (unsigned i = 0; i < N; i++) {
237         tables[n][i] = new double[1 « (K + 1)];
238         links[n][i] = new unsigned[K + 1];
239       }
240     }
241   }
242
243   /*********************************************
244    *
245    * Load the epistasis links from file
246    *
247    * @param _file open file of the instance
248    *
249    *********************************************/
250   void loadLinks(std::fstream &_file) {
251     unsigned n, i, j;
252
253     for (i = 0; i < N; i++)
254       for (j = 0; j < K + 1; j++)
255         for (n = 0; n < M; n++)
256           _file » links[n][i][j];
257   }
258
259   /*********************************************
260    *
261    * Load the tables of contribution
262    *
263    * @param _file open file of the instance
264    *
265    *********************************************/
266   void loadTables(std::fstream &_file) {
267     unsigned n, i;
268     int j;
269
270     for (i = 0; i < N; i++)
271       for (j = 0; j < (1 « (K + 1)); j++)
272         for (n = 0; n < M; n++)
273           _file » tables[n][i][j];
274   }
275
276   /*********************************************
277    *
278    * Fitness function of a single-objective NK-landscapes
279    *
280    * @param _numObj the objective fuction to consider
281    * @param _sol the solution to evaluate
282    *
283    *********************************************/
284   double evalNK(unsigned _numObj, std::vector<bool> &_sol) {
285     double accu = 0.0;
286
287     for (unsigned int i = 0; i < N; i++)
288       accu += tables[_numObj][i][sigma(_numObj, _sol, (int)i)];
289
290     return accu / (double)N;
291   }
292
293   /*********************************************
294    *
295    * Extract epistatic links of the fitness contribution i
296    *
297    * @param _numObj the objective function to consider
298    * @param _sol the solution to evaluate
299    * @param _i bit of the contribution
300    *
301    * *********************************************/
302   unsigned int sigma(unsigned _numObj, std::vector<bool> &_sol, int _i) {
303     unsigned int n = 1;
304     unsigned int accu = 0;
305
306     for (unsigned int j = 0; j < K + 1; j++) {
307       if (_sol[links[_numObj][_i][j]] == 1)
308         accu = accu | n;
309
```

```
310       n = n « 1;
311     }
312
313     return accu;
314   }
315 };
316
317 }  // namespace pmnk
318 #endif  // __rMNKEval
```

5.11 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/solution.hpp File Reference

Implementation of a solution class.

```
#include <cassert>
#include <random>
#include "rMNKEval.hpp"
```

Classes

- class pmnk::Solution

  Standard solution class.
- class pmnk::GASolution

  Genetic algorithm solution wrapper (adds a fitness attribute to the Solution class)

Typedefs

- using **pmnk::DecisionVector** = std::vector< bool >
- using **pmnk::ObjectiveVector** = std::vector< double >

Enumerations

- enum class **DominanceType** { **DOMINATES** , **EQUAL** , **DOMINATED** , **INCOMPARABLE** }

5.11.1 Detailed Description

Implementation of a solution class.

Author

Pedro Rodrigues (pedror@student.dei.uc.pt)

Alexandre Jesus (ajesus@dei.uc.pt)

Version

0.1.0

Date

13-09-2021

Copyright

Copyright (c) 2021

5.12 solution.hpp

Go to the documentation of this file.

```
1
12 #ifndef SOLUTION_HPP
13 #define SOLUTION_HPP
14
15 #include <cassert>
16 #include <random>
17
18 #include "rMNKEval.hpp"
19
20 namespace pmnk {
21
22 using DecisionVector = std::vector<bool>;
23 using ObjectiveVector = std::vector<double>;
24
25 enum class DominanceType { DOMINATES, EQUAL, DOMINATED, INCOMPARABLE };
26
28 class Solution {
29  protected:
30   DecisionVector m_decision;
31   ObjectiveVector m_objective;
32
33  public:
34   explicit Solution() = default;
35
41   Solution(Solution const &other) = default;
42
48   Solution(Solution &&other) = default;
49
56   Solution &operator=(Solution const &other) = default;
57
64   Solution &operator=(Solution &&other) = default;
65
72   Solution(RMNKEval &rmnk, DecisionVector const &decision)
73       : m_decision(decision) {
74     eval(rmnk);
75   }
76
83   Solution(RMNKEval &rmnk, DecisionVector &&decision)
84       : m_decision(std::move(decision)) {
85     eval(rmnk);
86   }
87
94   DecisionVector const &decision_vector() const {
95     return m_decision;
96   }
97
104   ObjectiveVector const &objective_vector() const {
105     return m_objective;
106   }
107
113   std::size_t size() const noexcept {
114     return m_decision.size();
115   };
116
124   DominanceType dominance(Solution const &solution) const {
125     assert(m_decision.size() == solution.m_decision.size());
126
127     DominanceType res = DominanceType::EQUAL;
128     for (decltype(m_objective.size()) i = 0; i < m_objective.size(); ++i) {
129       if (m_objective[i] < solution.m_objective[i]) {
130         if (res == DominanceType::DOMINATES) {
131           return DominanceType::INCOMPARABLE;
132         }
133         res = DominanceType::DOMINATED;
134       } else if (m_objective[i] > solution.m_objective[i]) {
135         if (res == DominanceType::DOMINATED) {
136           return DominanceType::INCOMPARABLE;
137         }
138         res = DominanceType::DOMINATES;
139       }
140     }
141     return res;
142   }
143
153   friend std::ostream &operator«(std::ostream &os, Solution const &solution) {
154     for (auto const &i : solution.objective_vector()) {
155       os « i « " ";
156     }
157     os « "\n";
158     return os;
159   }
160
```

```
170   DecisionVector::reference operator[](std::size_t const i) {
171     return m_decision[i];
172   }
173
182   void eval(RMNKEval &rmnk) {
183     rmnk.eval(m_decision, m_objective);
184   }
185
194   template <typename RNG>
195   static Solution random_solution(RMNKEval &eval, RNG &generator) {
196     std::uniform_int_distribution<int> distrib(0, 1);
197
198     DecisionVector decision_vector(eval.getN(), 0);
199     for (std::size_t i = 0; i < decision_vector.size(); ++i)
200       decision_vector[i] = distrib(generator);
201
202     return Solution(eval, std::move(decision_vector));
203   }
204
217   template <typename RNG>
218   static Solution uniform_bit_flip_solution(RMNKEval &eval, RNG &generator,
219                                             Solution const &original) {
220     std::bernoulli_distribution distrib(1 / (double)original.decision_vector().size());
221
222     DecisionVector flipped = original.decision_vector();
223     for (decltype(flipped.size()) i = 0; i < flipped.size(); ++i) {
224       if (distrib(generator)) {
225         flipped[i] = !flipped[i];
226       }
227     }
228     return Solution(eval, std::move(flipped));
229   }
230
240   static std::vector<Solution> neighborhood_solutions(RMNKEval &eval, Solution const &original) {
241     std::vector<Solution> neighborhood;
242     neighborhood.reserve(original.decision_vector().size());
243
244     for (size_t i = 0; i < original.decision_vector().size(); ++i) {
245       auto decision_vector = original.decision_vector();
246       decision_vector[i] = !decision_vector[i];
247
248       neighborhood.emplace_back(eval, std::move(decision_vector));
249     }
250
251     for (size_t i = 0; i < original.decision_vector().size(); ++i) {
252       for (size_t j = i + 1; j < original.decision_vector().size(); ++j) {
253         if (original.decision_vector()[i] == original.decision_vector()[j]) {
254           continue;
255         }
256
257         auto decision_vector = original.decision_vector();
258         swap(decision_vector[i], decision_vector[j]);
259         neighborhood.emplace_back(eval, std::move(decision_vector));
260       }
261     }
262     return neighborhood;
263   }
264 };
265
267 class GASolution : public Solution {
268   double m_fitness;
269
270  public:
271
276   explicit GASolution() = default;
277
284   GASolution(Solution &&sol, double fitness)
285       : Solution(std::forward<Solution>(sol))
286       , m_fitness(fitness) {}
287
294   GASolution(Solution &&sol)
295       : GASolution(std::forward<Solution>(sol), 0) {}
296
302   [[nodiscard]] constexpr double const &fitness() const {
303     return m_fitness;
304   }
305
311   void set_objv(ObjectiveVector &&objv) {
312     m_objective = std::move(objv);
313   }
314
320   constexpr void set_fitness(double const fitness) {
321     m_fitness = fitness;
322   }
323 };
324 }  // namespace pmnk
325 #endif  // SOLUTION_HPP
```

5.13 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/Utils/utils.hpp File Reference

Project Utility functions.

```
#include "solution.hpp"
```

Functions

- template<typename Vec , typename S >
  bool pmnk::add_non_dominated (Vec &solutions, S &&solution)
  Utility function resposible for maintaining a container of non-dominated solutions.

5.13.1 Detailed Description

Project Utility functions.

Author

Pedro Rodrigues (pedror@student.dei.uc.pt)
Alexandre Jesus (ajesus@dei.uc.pt)

Version

0.1.0

Date

13-09-2021

Copyright

Copyright (c) 2021

5.13.2 Function Documentation

5.13.2.1 add_non_dominated()

```
template<typename Vec , typename S >
bool pmnk::add_non_dominated (
            Vec & solutions,
            S && solution )
```

Utility function resposible for maintaining a container of non-dominated solutions.

Template Parameters

Vec	The type for the container holding the solutions.
S	The type for a solution to be added to the solution container.

Parameters

solutions	The container for the non-dominated solutions.
solution	The solution to be added to the container.

Returns

true If the solution was successfully added to the container

false If the solution to be added is dominated by other solution already present and failed to be inserted.

5.14 utils.hpp

Go to the documentation of this file.

```
1
13 #ifndef UTILS_HPP
14 #define UTILS_HPP
15
16 #include "solution.hpp"
17
18 namespace pmnk {
19
32 template <typename Vec, typename S>
33 bool add_non_dominated(Vec &solutions, S &&solution) {
34   for (std::size_t i = 0; i < solutions.size();) {
35     auto d = solution.dominance(solutions[i]);
36     if (d == DominanceType::EQUAL) {
37       if (solution.decision_vector() == solutions[i].decision_vector()) {
38         return false;
39       } else {
40         for (++i; i < solutions.size(); ++i) {
41           if (solution.decision_vector() == solutions[i].decision_vector()) {
42             return false;
43           }
44         }
45         break;
46       }
47     } else if (d == DominanceType::DOMINATES) {
48       solutions[i] = std::move(solutions.back());
49       solutions.pop_back();
50     } else if (d == DominanceType::DOMINATED) {
51       return false;
52     } else {
53       ++i;
54     }
55   }
56   solutions.push_back(std::forward<S>(solution));
57   return true;
58 }
59 }  // namespace pmnk
60 #endif  // UTILS_HPP
```

5.15 /home/pedro/Documents/projects/anytime-pmnk-landscapes/src/↩ Utils/wfg.hpp File Reference

Implementation of the wfg algorithm to calculate hypervolumes.

```
#include <algorithm>
#include <array>
#include <limits>
#include <type_traits>
#include <vector>
```

Classes

- class pmnk::hvobj< T >

  Implementation of an API that supports among others, set/point hypervolume calculations (using the WFG algorithm)

Functions

- template<typename T >
  auto pmnk::weakly_dominates (T const &lhs, T const &rhs)

  Check pareto weakly dominance between two points.
- template<typename T >
  void pmnk::insert_non_dominated (T &&sol, std::vector< T > &set)

  Insert a point (solution) into a non dominated (solution) set of points.
- template<typename Iter , typename T >
  auto pmnk::limit_set (Iter begin, Iter end, T const &sol)

  Limit the set of points by replacing them with the points whose value in each objective is limited to be no better than the contributing point.
- template<typename T , typename R >
  auto pmnk::point_hv (T const &p, R const &r)

  Compute hypervolume of a point in relation to a reference.
- template<typename S , typename T >
  std::common_type_t< typename S::value_type::value_type, typename T::value_type > pmnk::set_hv_wfg (S const &s, T const &ref)

  Compute a set hypervolume value given a reference point using the wfg algorithm. (Worker function)
- template<typename S , typename T >
  auto pmnk::set_hv (S const &s, T const &ref)

  Compute a set hypervolume value given a reference point using the wfg algorithm. (Wrapper function)
- template<typename T , typename S , typename R >
  auto pmnk::point_hvc (T const &p, S const &s, R const &ref)

  Calculate a point hypervolume contribution to a set.

5.15.1 Detailed Description

Implementation of the wfg algorithm to calculate hypervolumes.

Author

Alexandre Jesus (ajesus@dei.uc.pt)

Date

13-09-2021

Copyright

5.15.2 Function Documentation

5.15.2.1 insert_non_dominated()

```
template<typename T >
void pmnk::insert_non_dominated (
             T && sol,
             std::vector< T > & set )
```

Insert a point (solution) into a non dominated (solution) set of points.

Template Parameters

T	The type for the point being supplied as a parameter.

Parameters

sol	The point (solution) to be added to the set
set	The (solution) set of non dominated points (solutions)

5.15.2.2 limit_set()

```
template<typename Iter , typename T >
auto pmnk::limit_set (
             Iter begin,
             Iter end,
             T const & sol )
```

Limit the set of points by replacing them with the points whose value in each objective is limited to be no better than the contributing point.

Template Parameters

Iter	The type for and iterator of a set of points
T	The type for the contributing point being supplied as a parameter

Parameters

begin	An iterator for the begining of the set
end	An iterator for the end of the set
sol	The contributing solution

Returns

auto The limited set

5.15.2.3 point_hv()

```
template<typename T , typename R >
auto pmnk::point_hv (
            T const & p,
            R const & r )
```

Compute hypervolume of a point in relation to a reference.

Template Parameters

T	The type for the point being supplied as a parameter
R	The type for the reference being supplied as a parameter

Parameters

p	The point whose hypervolume value is to be computed
r	The reference point.

Returns

auto The hypervolume value.

5.15.2.4 point_hvc()

```
template<typename T , typename S , typename R >
auto pmnk::point_hvc (
            T const & p,
            S const & s,
            R const & ref )
```

Calculate a point hypervolume contribution to a set.

Template Parameters

S	The type for the set supplied as a parameter.
T	The type for the point being supplied as a parameter.
R	The type for the reference point supplied as a parameter.

Parameters

p	The point whose contribution is going to calculated.
s	The solution set.
ref	The reference point.

Returns

auto The point hypervolume contribution.

5.15.2.5 set_hv()

```
template<typename S , typename T >
auto pmnk::set_hv (
            S const & s,
            T const & ref )
```

Compute a set hypervolume value given a reference point using the wfg algorithm. (Wrapper function)

Template Parameters

S	The type for the set supplied as a parameter
T	The type for the point being supplied as a parameter

Parameters

s	The solution set
ref	The reference point.

Returns

std::common_type_t<typename S::value_type::value_type, typename T::value_type> The resulting set hypervolume value.

5.15.2.6 set_hv_wfg()

```
template<typename S , typename T >
std::common_type_t< typename S::value_type::value_type, typename T::value_type > pmnk::set_↩
hv_wfg (
            S const & s,
            T const & ref )
```

Compute a set hypervolume value given a reference point using the wfg algorithm. (Worker function)

Template Parameters

S	The type for the set supplied as a parameter
T	The type for the point being supplied as a parameter

Parameters

s	The solution set
ref	The reference point.

Returns

std::common_type_t<typename S::value_type::value_type, typename T::value_type> The resulting set hypervolume value.

5.15.2.7 weakly_dominates()

```
template<typename T >
auto pmnk::weakly_dominates (
             T const & lhs,
             T const & rhs )
```

Check pareto weakly dominance between two points.

Template Parameters

T	The type for the point being supplied as a parameter.

Parameters

lhs	The first point (left hand side point)
lhs	The second point (left hand side point)

Returns

auto A boolean with the value true if the left hand side point (lhs) weakly dominated the right hand side point (rhs)

5.16 wfg.hpp

Go to the documentation of this file.

```
1
9 #ifndef WFG_H
10 #define WFG_H
11
12 #include <algorithm>
13 #include <array>
```

```cpp
#include <limits>
#include <type_traits>
#include <vector>

// This code assumes maximizing objective functions

namespace pmnk {

template <typename T>
auto weakly_dominates(T const& lhs, T const& rhs) {
  for (decltype(lhs.size()) i = 0; i < lhs.size(); ++i) {
    if (lhs[i] < rhs[i]) {
      return false;
    }
  }
  return true;
}

template <typename T>
void insert_non_dominated(T&& sol, std::vector<T>& set) {
  for (auto it = set.begin(); it != set.end(); ++it) {
    if (weakly_dominates(*it, sol)) {
      return;
    } else if (weakly_dominates(sol, *it)) {
      *it = std::move(set.back());
      set.pop_back();
      set.erase(
          std::remove_if(it, set.end(), [&sol](auto const& s) { return weakly_dominates(sol, s); }),
          set.end());
      break;
    }
  }
  set.push_back(std::move(sol));
}

template <typename Iter, typename T>
auto limit_set(Iter begin, Iter end, T const& sol) {
  std::vector<T> res;
  res.reserve(static_cast<typename std::vector<T>::size_type>(std::distance(begin, end)));
  for (; begin != end; ++begin) {
    auto aux = *begin;
    for (size_t i = 0; i < sol.size(); ++i) {
      aux[i] = std::min(aux[i], sol[i]);
    }
    insert_non_dominated(std::move(aux), res);
  }
  return res;
}

template <typename T, typename R>
auto point_hv(T const& p, R const& r) {
  auto res = p[0] - r[0];
  for (size_t i = 1; i < p.size(); ++i) {
    res *= p[i] - r[i];
  }
  return res;
}

template <typename S, typename T>
std::common_type_t<typename S::value_type::value_type, typename T::value_type> set_hv_wfg(
    S const& s, T const& ref) {
  using result_t = std::common_type_t<typename S::value_type::value_type, typename T::value_type>;
  auto res = result_t(0);
  for (auto it = s.begin(); it != s.end(); ++it) {
    res += point_hv(*it, ref) - set_hv_wfg(limit_set(std::next(it), s.end(), *it), ref);
  }
  return res;
}

template <typename S, typename T>
auto set_hv(S const& s, T const& ref) {
  std::vector<T> v;
  v.reserve(s.size());
  for (auto const& sol : s) {
    v.push_back(sol.objective_vector());
  }
  sort(v.begin(), v.end(), [](auto const& a, auto const& b) { return a[0] < b[0]; });
  return set_hv_wfg(v, ref);
}

template <typename T, typename S, typename R>
auto point_hvc(T const& p, S const& s, R const& ref) {
  std::vector<T> v;
  v.reserve(s.size());
  for (auto const& sol : s) {
    v.push_back(sol.objective_vector());
  }
```

```
172   sort(v.begin(), v.end(), [](auto const& a, auto const& b) { return a[0] < b[0]; });
173   return point_hv(p, ref) - set_hv_wfg(limit_set(v.begin(), v.end(), p), ref);
174 }
175
177 template <typename T>
178 class [[nodiscard]] hvobj {
179  public:
180   using hv_type = T;
181   using ovec_type = std::vector<hv_type>;
182   using set_type = std::vector<ovec_type>;
183
184   constexpr explicit hvobj(ovec_type const& r)
185       : m_hv(0)
186       , m_set()
187       , m_ref(r) {}
188
189   constexpr hvobj(hvobj const& other) = default;
190   constexpr hvobj(hvobj&& other) noexcept = default;
191
193   [[nodiscard]] constexpr auto value() const {
194     return m_hv;
195   }
196
198   template <typename V>
199   [[nodiscard]] constexpr auto contribution(V const& v) const {
200     return m_point_hv(v, m_ref) - m_set_hv(m_limit_set(m_set, v), m_ref);
201   }
202
204   template <typename V>
205   constexpr auto insert(V&& v) {
206     auto hvc = contribution(v);
207     if (hvc != 0) {
208       m_insert_non_dominated(std::forward<V>(v), m_set);
209       m_hv += hvc;
210     }
211     return hvc;
212   }
213
216   template <typename V>
217   constexpr auto remove(V const& v) {
218     auto it = std::find(m_set.begin(), m_set.end(), v);
219     if (it == m_set.end())
220       return -1.0;
221     m_set.erase(it);
222     auto hvc = contribution(v);
223     m_hv -= hvc;
224     return hvc;
225   }
226
227  private:
229   template <typename V>
230   [[nodiscard]] constexpr auto m_weakly_dominates(V const& a, V const& b) const {
231     for (size_t i = 1; i < a.size(); ++i) {
232       if (a[i] < b[i]) {
233         return false;
234       }
235     }
236     return true;
237   }
238
240   template <typename V, typename C>
241   void m_insert_non_dominated(V&& v, C& set) const {
242     auto it = set.begin();
243
244     for (; it != set.end() && (*it)[0] > v[0]; ++it) {
245       if (m_weakly_dominates(*it, v)) {
246         return;
247       }
248     }
249
250     for (; it != set.end() && (*it)[0] == v[0]; ++it) {
251       if (m_weakly_dominates(*it, v)) {
252         return;
253       } else if (m_weakly_dominates(v, *it)) {
254         *it = std::forward<V>(v);
255         set.erase(std::remove_if(std::next(it), set.end(),
256                                  [this, it](auto const& a) { return m_weakly_dominates(*it, a); }),
257                   set.end());
258         return;
259       }
260     }
261
262     if (it == set.end()) {
263       set.push_back(std::forward<V>(v));
264     } else {
265       auto aux = std::forward<V>(v);
266       std::swap(aux, *it);
```

```
267       for (auto jt = std::next(it); jt != set.end(); ++jt) {
268         if (m_weakly_dominates(*it, aux)) {
269           set.erase(
270               std::remove_if(jt, set.end(),
271                              [this, it](auto const& a) { return m_weakly_dominates(*it, a); }),
272               set.end());
273           return;
274         } else {
275           std::swap(aux, *jt);
276         }
277       }
278       if (!m_weakly_dominates(*it, aux)) {
279         set.push_back(std::move(aux));
280       }
281     }
282   }
283
284   // limit_set (implementation)
285   template <typename S, typename V>
286   auto m_limit_set(S const& s, V const& v) const {
287     S res;
288     res.reserve(s.size());
289     for (auto const& p : s) {
290       auto aux = p;
291       for (size_t i = 0; i < aux.size(); ++i) {
292         aux[i] = std::min(aux[i], v[i]);
293       }
294       m_insert_non_dominated(std::move(aux), res);
295     }
296     return res;
297   }
298
300   template <typename V, typename R>
301   auto m_point_hv(V const& v, R const& r) const {
302     auto res = v[0] - r[0];
303     for (size_t i = 1; i < v.size(); ++i) {
304       res *= v[i] - r[i];
305     }
306     return res;
307   }
308
310   template <typename S, typename R>
311   auto m_set_hv3d(S const& s, R const& r) const {
312     using array2_t = std::array<hv_type, 2>;
313
314     auto aux = std::vector<array2_t>{{r[1], std::numeric_limits<hv_type>::max()},
315                                      {std::numeric_limits<hv_type>::max(), r[2]}};
316
317     hv_type v = 0;
318     hv_type a = 0;
319     hv_type z = 0;
320
321     for (auto const& p : s) {
322       v += a * (z - p[0]);
323       z = p[0];
324
325       auto tmp = array2_t{p[1], p[2]};
326       auto it = std::lower_bound(aux.begin(), aux.end(), tmp,
327                                  [](auto const& a, auto const& b) { return a[1] > b[1]; });
328       auto jt = it;
329
330       auto r0 = (*std::prev(it))[0];
331       auto r1 = tmp[1];
332       for (; (*it)[0] <= tmp[0]; ++it) {
333         a += (tmp[0] - r0) * (r1 - (*it)[1]);
334         r0 = (*it)[0];
335         r1 = (*it)[1];
336       }
337       a += (tmp[0] - r0) * (r1 - (*it)[1]);
338       if (jt != it) {
339         *jt = tmp;
340         aux.erase(++jt, it);
341       } else {
342         aux.insert(it, tmp);
343       }
344     }
345     v += a * (z - r[0]);
346     return v;
347   }
348
350   template <typename S, typename R>
351   auto m_set_hv(S const& s, R const& r, hv_type c = 1) const -> hv_type {
352     hv_type v = 0;
353
354     if (s.size() == 0) {
355       return 0;
356     }
```

```
357
358     if (s.begin()->size() == 2) {
359       hv_type r1 = r[1];
360       for (auto const& p : s) {
361         v += (p[1] - r1) * (p[0] - r[0]);
362         r1 = p[1];
363       }
364       v *= c;
365     } else if (s.begin()->size() == 3) {
366       v = c * m_set_hv3d(s, r);
367     } else {
368       auto newr = std::vector<hv_type>();
369       for (size_t i = 1; i < r.size(); ++i) {
370         newr.push_back(r[i]);
371       }
372       auto newl = std::vector<std::vector<hv_type»();
373       newl.reserve(s.size());
374       for (auto const& p : s) {
375         auto newc = c * (p[0] - r[0]);
376         auto newp = std::vector<hv_type>();
377         for (size_t i = 1; i < p.size(); ++i) {
378           newp.push_back(p[i]);
379         }
380
381         v += newc * m_point_hv(newp, newr) - m_set_hv(m_limit_set(newl, newp), newr, newc);
382
383         m_insert_non_dominated(std::move(newp), newl);
384       }
385     }
386     return v;
387   }
388
389   hv_type m_hv;
390   set_type m_set;
391   ovec_type m_ref;
392 };
393
394 }  // namespace pmnk
395 #endif  // WFG_H
```

Index